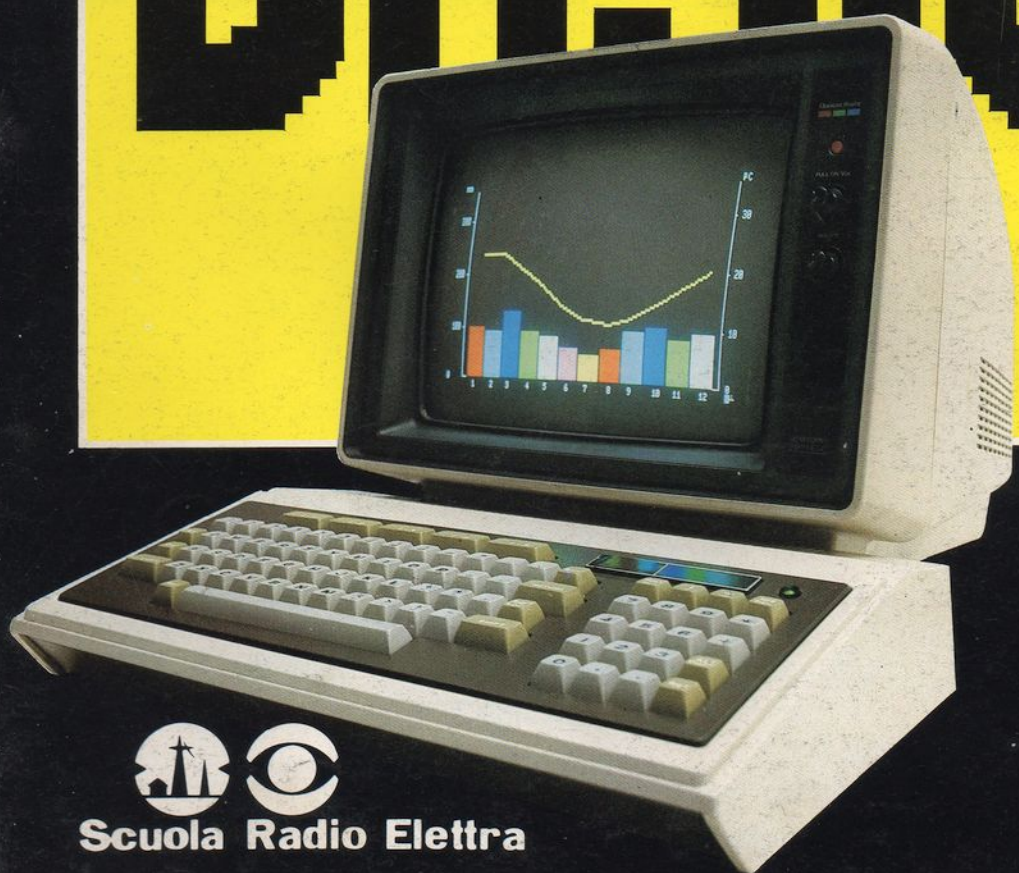
PARLA BASIC



Scuola Radio Elettra

Scuola Radio Elettra
Torino

Analisi e programmazione

Lezione 5

1 — Costanti e variabili numeriche
2 — Costanti e variabili stringa
3 — Rappresentazione interna ed esterna delle variabili
4 — Le espressioni algebriche
5 — Qualche dettaglio in piú sull'istruzione **IF... THEN...**
6 — Comandi BASIC
7 — Comandi di utilità
8 — Istruzioni e comandi per la raffigurazione grafica

Che cosa apprenderà

In questa lezione saranno trattati in modo dettagliato i valori su cui operano le istruzioni del linguaggio BASIC, ossia le costanti e le variabili (numeriche e stringa).

Si proseguirà inoltre con l'esame dei comandi per la gestione dell'unità a disco e delle periferiche piú comuni. Infine verranno presi in considerazione i comandi e le istruzioni per la raffigurazione grafica automatica su monitor, stampante, plotter, ecc.

1. - Costanti e variabili numeriche

Come precedentemente precisato, quando si batte un carattere sulla tastiera di un personal computer, speciali circuiti elettronici provvedono a codificare il carattere (che può essere una cifra decimale, una lettera alfabetica o un altro carattere grafico qualsiasi) in un particolare codice binario, chiamato ASCII (già riportato nella lezione *Analisi e Programmazione 2*).

In tale codice ciascun carattere particolare battuto sulla tastiera è rappresentato da un gruppo di 8 bit, che viene immesso nella memoria principale del computer (*fig. 1*), in una determinata zona denominata **cella elementare** o **byte**.

Fig. 1 - Codificazione dei caratteri battuti sulla tastiera.

Ogni carattere, o cifra, o simbolo è quindi contenuto in una cella elementare di memoria o byte ed è codificato in codice ASCII binario.

Per memorizzare nella memoria principale i numeri (insiemi o gruppi di cifre) e le parole (insiemi o gruppi di caratteri alfabetici), non basta però premere semplicemente dei tasti.

Affinché i numeri e le parole giungano veramente nella memoria e vi restino memorizzati, occorre che siano dotati di un nome o etichetta: il computer in questo senso si comporta meglio dell'uomo, perché usa chiamare per nome!

Quando un numero particolare viene usato nel calcolo algebrico, esso viene prelevato da particolari zone di memoria dove è riconoscibile e rintracciabile soltanto se ha un nome.

Si è anche detto nelle precedenti lezioni che la memoria del computer si comporta come una cassettiera: ogni cassetto costituisce una zona di memoria (generalmente formata da un gruppo di byte) dove risiede un dato numerico o alfabetico; lo sportello con l'etichetta sopra fa da facciata e serve per il riconoscimento del contenuto del cassetto (*fig. 2-a*).

Come si fa in linguaggio BASIC ad assegnare una etichetta o un nome ad un valore, in modo che questo possa essere richiamato, quante volte si desidera, nel corso dell'elaborazione?

Si utilizza l'istruzione di assegnazione **LET** studiata nelle lezioni precedenti, oppure si usa scrivere una semplice equazione che funge da assegnazione.

Assegnazione dei nomi alle variabili o alle costanti

Per esempio, se si vuole assegnare il nome PI (iniziali di pi-greco=3,14159) al numero 3,14159 reale (noto come rapporto fisso tra una circonferenza ed il suo diametro), si può procedere all'assegnazione in due modi equivalenti, cioè scrivere:

```
10 LET PI=3.14159
```

oppure:

```
10 PI=3.14159
```

Il computer, eseguendo una di queste due istruzioni, immetterà nel cassetto di memoria di nome PI la costante 3,14159, cosicché, quando il programma dovrà utilizzare questa costante per calcolare la lunghezza di una circonferenza (anche questa dotata di nome, per esempio LC), si potrà scrivere il calcolo in questo modo:

```
80 LC=2*PI*R
```

chiamando i numeri con il loro proprio nome.

Come si vede nell'esempio considerato (istruzione 8Ø), anche ad R (raggio del cerchio) è stato precedentemente assegnato un valore; quindi, al momento dell'esecuzione del calcolo, il computer è in grado di operare sui numeri prelevandoli o depositandoli nel cassetto di memoria opportuno (*fig. 2-b*).

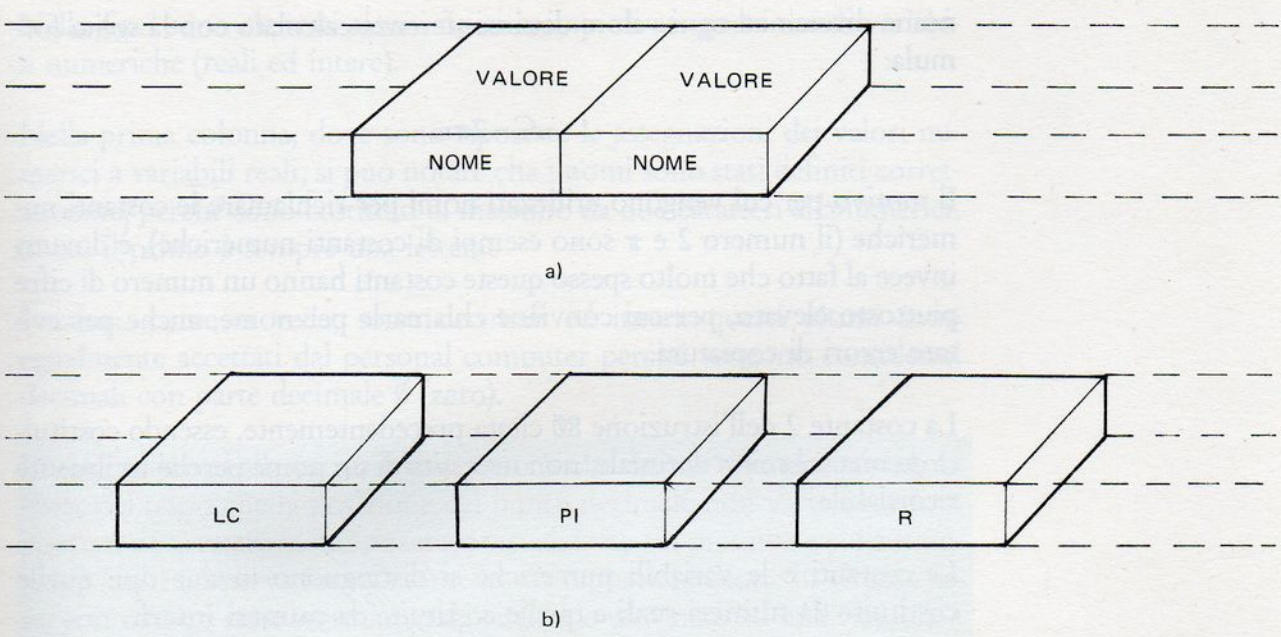


Fig. 2 – La memoria del computer è come una "cassettiera" d'informazioni.

Da quanto precisato prima si può altresí rilevare che, quando l'istruzione **LET** tratta costanti o variabili numeriche, essa ordina al computer di calcolare il valore dell'espressione posta a destra del segno di uguale (2*PI*R) e di attribuire tale valore alla variabile (LC) scritta a sinistra del segno di uguale.

Quando, durante lo svolgimento del programma, occorre cambiare il valore (o contenuto) ad una certa variabile, è sufficiente eseguire un'altra istruzione di assegnazione (usando o meno la parola LET, come precedentemente visto).

Perché nella programmazione si fa sempre uso dei nomi per richiamare le variabili e le costanti, almeno tutte le volte che ciò è possibile?

Per quanto riguarda le variabili, ciò è dovuto al tipo stesso di queste entità matematiche. Infatti **le variabili**, durante l'elaborazione, **rappresentano valori che sono soggetti a continua variazione, per cui si può dire che è quasi indispensabile richiamarle con un nome fisso.**

Ad esempio, la variabile LC dell'istruzione 8Ø vista prima è un numero che varia in funzione o dipendenza del valore di R. Se si introducono piú valori di R, si ottengono piú valori di LC e sarebbe illogico riservare un

nome diverso ad ogni valore di circonferenza calcolato con la solita formula:

$$C=2\pi r.$$

Il motivo per cui vengono utilizzati nomi per richiamare le costanti numeriche (il numero 2 e π sono esempi di costanti numeriche), è dovuto invece al fatto che molto spesso queste costanti hanno un numero di cifre piuttosto elevato, per cui conviene chiamarle per nome, anche per evitare errori di copiatura.

La costante 2 dell'istruzione 8Ø citata precedentemente, essendo costituita da una sola cifra decimale, non necessita di un nome perché facilmente ricordabile.

Le costanti e le variabili numeriche si distinguono in due tipi: quelle costituite da **numeri reali** e quelle costituite da **numeri interi.**

Da un punto di vista matematico questa distinzione non è del tutto esatta, perché i numeri interi sono un sottoinsieme dell'insieme dei numeri reali (cioè i numeri interi sono anche numeri reali).

In informatica però i numeri reali vengono rappresentati in un modo diverso dai numeri interi.

Intanto i nomi delle variabili reali si differenziano dai nomi delle variabili intere per il fatto che queste ultime riportano dopo i primi due caratteri alfanumerici il simbolo particolare %, mentre le variabili reali non hanno tale simbolo.

Variabili reali	**Variabili intere**
A1=1.41	A%=1Ø
AB=+44.ØØ3	MN%=-1
CD=Ø.Ø1	CC%=3ØØØØ
H=33.Ø1	PC%=2
TS=5	SS%=32767
X=-1Ø	NS%=-32768
I=1	HP%=33ØØØ
R=Ø.ØØ1	HS%=-32ØØØ
Y=9999999999	R%=3.5
R1=-Ø.Ø1	X9%=33.9
Y9=-999999999	XØ%=4.1

Fig. 3 - Esempi di assegnazione di variabili.

Nella *fig. 3* sono elencati alcuni tipici esempi di assegnazione di variabili numeriche (reali ed intere).

Nella prima colonna, dove sono riportate le assegnazioni dei valori numerici a variabili reali, si può notare che i nomi sono stati definiti correttamente, perché sono costituiti al massimo da due caratteri alfanumerici, di cui il primo è sempre una lettera.

I valori, come si vede, sono decimali ed interi: questi ultimi sono egualmente accettati dal personal computer perché vengono considerati decimali con parte decimale Ø (zero).

I valori assegnati alle variabili reali sono anche definiti **numeri in virgola fissa**, nel senso che la posizione del punto decimale non varia quando il numero deve essere memorizzato o visualizzato sul monitor o sulla stampante (ossia conserva formalmente la sua rappresentazione in virgola fissa, cosí come è stato introdotto da tastiera o assegnato ad una variabile); ciò vale però se il valore numerico è compreso nell'intervallo

$$| \text{Ø.Ø1} | \div | \text{999999999.9} |$$

dove con il simbolo | numero | si intende il valore assoluto del numero stesso (il valore assoluto di un numero positivo o negativo non è altro che il numero privo di segno).

Quindi valori numerici in virgola fissa quali i seguenti:

+33.5; -4.Ø1; -Ø.Ø1; +9999.999; -99.999;
Ø.Ø1; Ø.Ø1567; -Ø.8956; +1Ø5.995

essendo tutti in valore assoluto compresi nell'intervallo

Intervallo di validità per la rappresentazione in virgola fissa

$$| \text{Ø.Ø1} | \div | \text{999999999.9} |$$

conservano formalmente la stessa rappresentazione in virgola fissa con cui sono stati introdotti.

I valori numerici reali che invece sono fuori di tale intervallo subiscono una trasformazione formale, chiamata **notazione esponenziale.**

Rappresentazione in "notazione esponenziale"

La notazione esponenziale dei numeri consiste nel trasformarne il valore formalmente a guisa di prodotto di una mantissa (parte piú significativa del numero) per una potenza di 1Ø, in modo che il valore resti invariato.

Per esempio, il numero 1Ø1.58 in notazione esponenziale si può scrivere come il prodotto:

$$1.Ø158 \times 10^2 \qquad (10^2 = 100)$$

oppure come il prodotto:

$$Ø.1Ø158 \times 10^3 \qquad (10^3 = 1000)$$

Infatti, se si eseguono questi due prodotti, si ottiene effettivamente il numero originario 1Ø1.58.

Per giungere alla vera notazione esponenziale, il linguaggio BASIC sostituisce il numero 1Ø (base della potenza) con la lettera E, allo scopo di indicare che il numero è rappresentato in notazione esponenziale, cosicché il numero 1Ø1.58 in notazione esponenziale si scrive:

$$1.Ø158E Ø2 \qquad (E\ Ø2 = 10^2)$$

Ed il numero -Ø.ØØ589 come si scrive?

In questo caso, oltre ad avere il segno negativo, il numero è in valore assoluto minore di 1.

Esso si può scrivere in forma matematicamente corretta come segue:

$$-5.89 \times 10^{-3} \qquad (10^{-3} = 0{,}001)$$

ed in notazione esponenziale BASIC nel seguente modo:

$$-5.89E-Ø3 \qquad (E-Ø3 = 10^{-3})$$

Nel linguaggio BASIC, se si introduce nella variabile di nome A il seguente numero espresso in virgola fissa:

```
LET A=-Ø.ØØØØ85
```

e poi si visualizza questo valore sullo schermo o sulla stampante con l'istruzione immediata:

```
PRINT A
```

si vedrà apparire il numero precedente, scritto nella seguente forma

esponenziale:

-8.5E-Ø5 (E-Ø5 = 10^{-5})

Se si introduce in A (sempre con l'istruzione **LET** di assegnazione) il numero 9999999999, cioè:

LET A=9999999999

e si stampa

PRINT A,

si vedrà apparire il seguente valore arrotondato:

1. E 1Ø

(infatti il numero 9999999999 = Ø.9999999999·10^{10} = 1·10^{10}).

Se invece si introduce in A il valore -33.Ø55, si vedrà apparire il numero in notazione virgola fissa cosí come è stato introdotto, cioè comparirà:

-33.Ø55.

Rivedendo gli esempi di assegnazione di variabili fatti prima, si può notare che ad alcune variabili reali (come R, Y) sono stati assegnati valori fuori dell'intervallo

| Ø.Ø1 | ÷ | 999999999.9 | ;

se queste variabili vengono stampate, appariranno espresse in notazione esponenziale.

Intervallo di validità per la rappresentazione "intera"

Per quanto riguarda i numeri interi, questi conservano la forma intera se il loro valore è compreso in quest'altro intervallo:

-32768 ÷ +32767.

Le variabili intere che sono state indicate con HP%, HS% nella *fig. 3* sono quindi formalmente esatte come nome, ma errate come valore, perché ad esse sono stati assegnati numeri interi fuori dell'intervallo:

-32768 ÷ +32767.

Il linguaggio BASIC emanerebbe un messaggio di errore per queste variabili intere.

Nella stessa *fig. 3* sono state riportate alcune assegnazioni di variabili intere (R%=3.5; X9%=33.9; XØ%=4.1) che formalmente sembrano errate, ma che il sistema accetta ugualmente considerando i valori senza la parte decimale; cioè, alle tre variabili intere R%, X9%, XØ% vengono assegnati i seguenti valori interi:

```
R%=3
X9%=33
XØ%=4.
```

In BASIC esiste comunque un'istruzione particolare, che assegna ad una variabile intera la parte intera di un numero decimale; essa è l'istruzione:

```
INT (X),
```

dove X è un numero decimale qualsiasi.

Per esempio, l'istruzione:

```
A%=INT (44.7Ø)
```

assegna il numero intero 44 alla variabile intera A%.

2. - Costanti e variabili stringa

Con il termine **stringa** si intende, come è noto, un gruppo di caratteri alfanumerici, cioè un insieme di lettere, numeri, caratteri grafici, ecc.

Per esempio, l'espressione alfabetica "SCUOLA RADIO ELETTRA", piú volte usata in queste lezioni, è una stringa alfabetica; il nominativo di un cliente con il suo codice fiscale costituisce un altro esempio di stringa alfanumerica.

Mentre i numeri non sono mai racchiusi tra caratteri speciali, le stringhe hanno questa particolarità, ossia sono delimitate all'inizio ed alla fine dal

carattere virgolette (”); questa indicazione permette al computer di riconoscerle come tali.

Come già precedentemente precisato, il nome che viene usato per indicare le variabili stringa deve terminare con il carattere $.

Le stringhe si differenziano dai numeri anche per il fatto che non sono operative, cioè non possono essere elaborate algebricamente; non ha senso, infatti, in un computer eseguire l'operazione:

```
A="B"*"C"
```

oppure

```
C="H"/3.
```

Le operazioni con le stringhe

E' invece possibile con le stringhe effettuare confronti logici e concatenamenti (specialmente di addizioni alfabetiche).

Si potrà per esempio scrivere:

```
10 IF "B"<=B$ THEN 300
20.....
```

che significa: se (IF) la lettera "B", all'istruzione 1Ø, risulta minore o uguale (alfabeticamente parlando, se la lettera B precede la lettera contenuta nella prima posizione della stringa di nome B$), il controllo dell'elaborazione passa a 3ØØ.

Quindi, se la prima lettera di B$ contiene la lettera B oppure una lettera successiva dell'alfabeto linguistico internazionale, il controllo del programma passa effettivamente all'istruzione 3ØØ, altrimenti prosegue per l'istruzione 2Ø...

Il confronto logico tra stringhe può essere utile per riconoscere se il nominativo di una lista precede alfabeticamente un altro nominativo e quindi per effettuare l'ordinamento alfabetico della lista, come se si trattasse di un elenco telefonico.

Per quanto riguarda il valore di stringa, questo, come già è stato detto, può comprendere qualsiasi carattere alfanumerico, tranne le virgolette che sono utilizzate come delimitatori; il numero dei caratteri, compresi gli spazi, può arrivare al massimo a 256 nei personal computer dotati di

BASIC piú evoluti, mentre è notevolmente inferiore (circa 80 caratteri) nei microsistemi.

Due o piú stringhe si possono concatenare con l'operatore + (piú); per esempio, se la variabile stringa A$ contiene la parola "SCUOLA" e un'altra variabile stringa B$ contiene la parola "RADIO ELETTRA", si può generare un'altra variabile stringa che contenga le due variabili, come di seguito indicato:

```
⋮                    1 spazio
10 A$ ="SCUOLA "      ↓
12 B$ ="RADIO ELETTRA"
14 C$ =A$+B$
16 PRINT C$
⋮
```

Se si esegue con il comando **RUN** questo piccolo programma, si vedrà apparire sullo schermo la scritta:

SCUOLA RADIO ELETTRA

ottenuta con la composizione delle parole delle due stringhe: "SCUOLA " e "RADIO ELETTRA".

Quando si eseguono concatenamenti tra stringhe, è necessario **aggiungere in coda od all'inizio di una stringa uno o piú spazi**, per evitare che le varie parole appaiano "attaccate"; quindi, nell'esempio fatto, dopo la lettera A della parola "SCUOLA" o prima della lettera R della parola "RADIO....", occorre inserire uno o piú spazi.

Costanti stringa e variabili stringa

Le costanti stringa sono entità che conservano il valore ad esse assegnato per tutta la durata dell'elaborazione, mentre le variabili stringa sono entità che possono assumere diversi valori nel corso dell'elaborazione. Per esempio, la lettera B dell'istruzione:

```
10 IF "B" <=B$ THEN...
```

è una costante stringa (che però non ha nome), mentre la variabile B$ contiene espressioni stringa che variano continuamente nell'elaborazione.

La distinzione tra costante e variabile è soltanto formale in programmazione; essa serve al programmatore per riconoscere se un valore cambia o resta invariato.

Esempi di costanti numeriche sono il valore di pi-greco, cioè $\pi = 3.14159$ e la costante neperiana **e**=2.718.

Esempi di costanti stringhe sono le espressioni seguenti: "SCUOLA RADIO ELETTRA", "PROGRAMMA DI CALCOLO LUNGHEZZA CIRCONFERENZA", "INTRODUCI NUMERO", ecc.

Sia le variabili numeriche sia le variabili stringa consistono invece in valori parziali di elaborazione, quantità di accumulo, contatori, risultati finali d'elaborazione, nominativi presi da un elenco alfabetico di nomi, codici alfanumerici di parti in magazzino, ecc.

Nella tabella della *fig. 4* sono riportati alcuni esempi significativi di costanti e di variabili stringa.

Costanti stringa	**Variabili stringa**
1) NO$="NOME COGNOME"	A$=B$+C$
2) L$="SCUOLA RADIO ELETTRA"	S1$=SØ$+"A"
3) CI$="CITTA' DI TORINO"	S9$=H$+"3.14159"
4) F$="L'UOMO E' UN ANIMALE RAGIONEVOLE"	AL$="*"+"*"+S$
5) "3.14159 E' IL NUMERO PI-GRECO"	MS$="INTRODUCI"+R$
6) "IL NUMERO **e**=2,718 E' LA BASE DEI NUMERI NEPERIANI"	Z$=NOME$+COGNOME$
7) RR$="A"+"B"	Y$=" "+"+"+VAL$

Fig. 4 – Esempi validi di assegnazione di costanti e di variabili stringa.

Assegnazione dei nomi a variabili e costanti stringa

Come si può notare, le variabili stringa riportano nomi di variabili stringa a sinistra del segno = (uguale), mentre a destra di questo segno c'è sempre, per le variabili, il nome di un'altra variabile stringa od una combinazione concatenata di costanti e variabili stringhe, ma non ci sono mai soltanto delle costanti.

Il motivo è semplice: se nella parte destra ci fosse soltanto una costante stringa (che si riconoscerebbe dall'espressione racchiusa tra virgolette), alla variabile il cui nome è riportato a sinistra del segno = verrebbe assegnato un valore costante e non piú variabile.

Sempre nella colonna destra della *fig. 4* si può ancora notare che alla varia-

bile stringa Z$ sono assegnate le due stringhe di nome: NOME$+COGNOME$, le quali hanno un numero di caratteri alfanumerici, prima del simbolo $, superiore a due.

Il sistema di linguaggio BASIC accetta per i nomi delle variabili o delle costanti (sia numeriche, sia stringa) un numero di caratteri anche superiore a due, ma riconosce soltanto i primi due caratteri, cioè, nel caso dell'esempio fatto, interpreta la scrittura:

```
NOME $+COGNOME $
```

come se fosse:

```
NO $+CO $
```

Ritornando alla *fig. 4*, si può rilevare che nella prima colonna le espressioni 5) e 6) rappresentano due esempi di costanti stringa a cui non è stato assegnato alcun nome di stringa.

Quale effetto può avere nella programmazione scrivere delle costanti (sia stringa sia numeriche) senza nome?

La risposta è semplice: ogni volta che nel corso del programma queste costanti debbono essere elaborate, occorre riscriverle sempre allo stesso modo (lunghe o brevi che siano).

Se per esempio il numero 3.14159 ricorre piú volte nelle istruzioni di un programma, occorre che ogni volta sia scritto esattamente com'è e, se per un caso fortuito, esso viene battuto in modo erroneo, ciò si rifletterà inesorabilmente sulla correttezza dei risultati dell'elaborazione.

E' buona regola quindi fare sempre uso di nomi da assegnare ai numeri e alle stringhe, specialmente a quelli che si ripetono piú volte nel programma.

3. – Rappresentazione interna ed esterna delle variabili

Come si è visto all'inizio di questa lezione, le variabili numeriche e stringa sono memorizzate ognuna con il proprio nome nella memoria

principale del computer; come siano memorizzate ed in quale zona della memoria riservata all'utente non è dato sapere al programmatore, tanto piú che i valori delle variabili sono in codice binario, quindi difficilmente interpretabili ed utilizzabili.

E' importante unicamente avere un'idea piú o meno approssimativa del numero di byte da cui è costituita ogni variabile, in modo che il programmatore possa rendersi conto grossolanamente di quanto sia ampia la zona di memoria occupata complessivamente dai dati e dal programma.

Una variabile numerica reale semplice (non elemento di matrice o vettore) occupa 7 byte, cioè 7 celle elementari di memoria costituite da un numero di bit pari a $7 \times 8 = 56$.

Due di questi 7 byte sono riservati al nome della variabile ed i rimanenti cinque al valore numerico che è rappresentato in notazione esponenziale.

Si è già visto che cosa si intende per notazione esponenziale.

Per esempio, il numero 3,1415926 si può scrivere in notazione esponenziale:

+31415926+01.

E' questo un tipo di rappresentazione esponenziale interna (cioè nella memoria principale) molto sintetico, che fa a meno del punto decimale, in quanto il sistema implicitamente lo considera sempre alla sinistra della prima cifra della mantissa (3).

Per la visualizzazione esterna dei numeri in notazione esponenziale si usa una rappresentazione un po' diversa:

+3.1415926 E 00

che significa:

$3{,}1415926 \cdot 10^0$ $\qquad (10^0 = 1)$

Naturalmente, la rappresentazione esponenziale interna viene memorizzata nella memoria principale in caratteri binari (0 e 1) ed occupa, come si è detto, 5 byte (4 byte per la mantissa, compreso il segno del numero, 1 byte per l'esponente).

Anche le variabili intere e stringa occupano 7 byte.

Per le variabili intere 2 byte sono riservati al nome, 2 byte al valore intero (che, come è noto, deve essere compreso tra -32768 e +32767) ed i rimanenti 3 byte sono occupati da bit Ø.

Per le variabili stringa 2 byte sono riservati al nome, 1 byte per la lunghezza della stringa, 2 byte per il puntatore (indirizzo contenuto nella memoria centrale) della stringa ed i rimanenti 2 byte sono coperti da bit Ø.

Anche l'occupazione di memoria da parte delle variabili varia da un sistema all'altro di personal computer, come diversi sono i criteri di memorizzazione.

I sistemi che per esempio utilizzano fino a sei caratteri alfanumerici per i nomi delle variabili impiegano dei byte in piú per la loro memorizzazione.

Quanto è stato precisato per le variabili e per le costanti (sia numeriche sia stringa) è valido anche per gli elementi di matrice (variabili con indici).

Basta aggiungere che i nomi di questi ultimi, mentre l'occupazione di memoria riservata al loro valore è identica a quella delle variabili, occu-

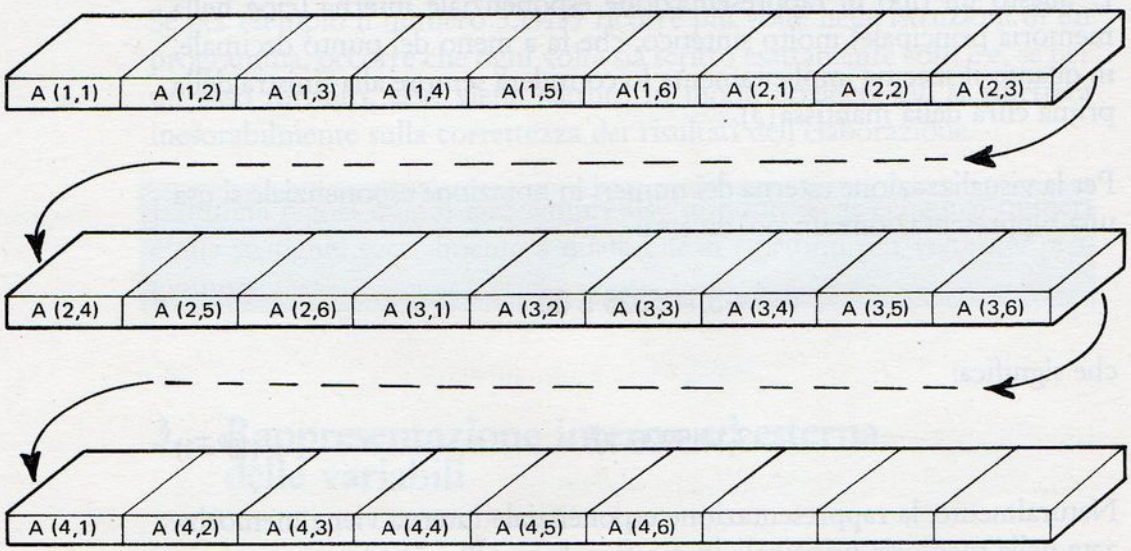


Fig. 5 - Memorizzazione della matrice A di 4 righe e 6 colonne.

pano invece piú byte.

Il motivo di ciò risiede nel fatto che gli elementi di matrice sono variabili dotate di indici che, come si vedrà in una prossima lezione, possono essere anche piú di due (se la matrice ha una sola dimensione si chiama anche vettore); questi indici fanno parte quindi di una struttura organizzata di dati (la matrice, appunto), che è memorizzata in memoria in modo sequenziale.

Nella *fig. 5* è rappresentato un esempio di memorizzazione di matrice di 4 righe e 6 colonne.

In tale figura ogni elemento di matrice, per essere individuato, ha bisogno di un nome, di un valore e di coordinate che ne consentano il reperimento immediato a partire da un indirizzo iniziale; ogni elemento quindi occupa una zona di memoria piú ampia di una variabile, circa 12 byte: 2 byte per il nome, 5 byte per il valore, 2 byte per entrambe le coordinate o indici dell'elemento, 1 byte per l'indirizzo dell'elemento concatenato successivo, 2 byte per lo spazio totale occupato dalla matrice intera.

A questo punto ci si può domandare: quanta memoria occupa un programma scritto in BASIC?

E' chiaro che l'area di memoria occupata dipende prima di tutto dal numero delle istruzioni immesse.

Ogni numero di istruzione, o linea di programma, occupa 2 byte di memoria, indipendentemente dal fatto che questi numeri abbiano il minimo valore (Ø) o il massimo valore (65535); la parola chiave richiede soltanto 1 byte di memoria, perché ad ognuna viene fatto corrispondere un codice numerico intero con valore compreso tra Ø e 128 (ed un numero intero compreso tra questi due estremi in binario non occupa piú di 8 bit, cioè non piú di 1 byte).

Tutti gli altri caratteri, che possono essere al massimo 239, riportati dopo la parola chiave, occupano 1 byte ciascuno.

Ogni loop (ciclo) attivo di programma che utilizza l'istruzione **FOR... NEXT** (di cui si parlerà dettagliatamente in una prossima lezione) occupa 16 byte di memoria e piú o meno altrettanti byte sono occupati dai risultati provvisori delle espressioni algebriche contenuti nelle parentesi.

Come si vede, il programmatore può farsi a priori un'idea di massima di quanta memoria ha bisogno un programma elaborato in linguaggio BASIC.

4. - Le espressioni algebriche

Nel linguaggio BASIC un'espressione algebrica è un insieme di valori numerici e di operatori aritmetici scritti in una certa sequenza dopo la parola chiave **PRINT**, oppure dopo il segno (=) di un'istruzione di assegnazione, oppure dopo il segno = di una funzione matematica.

In pratica, in BASIC, un'espressione algebrica si scrive come una normale espressione aritmetica, con alcune modifiche per quanto riguarda la rappresentazione dei valori numerici e degli operatori.

Espressione aritmetica	**Espressione BASIC**
1) $AREA=\pi R^2$	AR=3.14159*R↑2
2) $Y=2X+1$	Y=2*X+1
3) $Z=1+\left\{[1+(\text{sen}X+\cos Y)]^2\right\}$	Z=1+((1+(SIN(X)+COS(Y)))↑2)
4) $H=3\left\{1+\frac{2}{1+X}+\left[\frac{(X+Y)^2}{X-Y}\right]^3\right\}$	H=3*(1+2/(1+X)+((X+Y)↑2/(X-Y))↑3)
5) $S=S+\left\{1+[3+X]^2+Y^2\right\}$	S=S+(1+(3+X)↑2+Y↑2)
6) $\alpha=0,1$	AL=Ø.1
7) $\alpha=\beta+1$	ALFA=BETA+1
8) $X=\frac{Z+Y}{1+Z}$	X=(Z+Y)/(1+Z)
9) $X=\frac{(1+Y)^3}{(1-Y)^2}+1$	X=(1+Y)↑3/(1-Y)↑2+1
10) $Z=\sqrt{(1+X)^3+Y}+\alpha$	Z=SQR ((1+X)↑3+Y)+AL
11) $Y=(\text{arctg}X+1)^2+\text{sen}\frac{1}{X}$	Y=(ATN(X)+1)↑2+SIN(1/X)

Fig. 6 - Analogie tra espressioni aritmetiche ed espressioni BASIC.

Occorre premettere che gli spazi eventuali sono trascurati e che le parentesi quadre e graffe non esistono in BASIC; quando una parentesi ne racchiude un'altra, si raffigura sempre con una parentesi tonda, sicché un'espressione anche piuttosto complessa appare come una sequenza di numeri, di operatori e di parentesi tonde.

Nella *fig. 6* è riportato un prospetto in cui, a fianco di ogni espressione aritmetica, è raffigurata la corrispondente rappresentazione in linguaggio BASIC.

Esempi di calcolo algebrico e regole operative

L'esempio 1) della figura in questione rappresenta il calcolo dell'area di un cerchio di raggio R: come si può notare, il nome che si assegna alle costanti o ai risultati di un calcolo non può superare i due caratteri alfanumerici (AR al posto di AREA).

Il simbolo π (pi-greco) viene scritto con il suo proprio valore che, come è noto, è 3.14159.

Come già precisato, la moltiplicazione si indica con il carattere *, la divisione con il carattere /, l'elevazione a potenza con il carattere ↑ (in alcuni sistemi personal computer viene utilizzato il carattere ∧ oppure il doppio asterisco **), la somma e la sottrazione rispettivamente con i caratteri + e - .

Le denominazioni matematiche tipo α, β, γ (derivate dall'alfabeto greco) non esistono in BASIC; al loro posto vengono usati nomi di variabili reali come AL, BE, GA, ecc.

Delle funzioni trigonometriche e, in genere, delle principali funzioni matematiche si parlerà piú diffusamente in seguito; per il momento anticipiamo che la radice quadrata si scrive con il codice: **SQR** (X), dove X è l'argomento della radice e deve essere necessariamente un valore reale positivo, altrimenti viene emesso un messaggio di errore (d'altronde in matematica la radice di un numero negativo è un numero "immaginario").

L'uso delle parentesi tonde è molto importante nella programmazione elettronica; esse infatti tracciano la sequenza temporale del calcolo algebrico, indicano cioè se un'operazione è da effettuarsi prima oppure dopo un'altra.

La sintassi BASIC prevede un ordine di precedenza per l'esecuzione delle varie operazioni presenti in un'espressione algebrica; questo ordine è

riportato di seguito.

1° - Risoluzione dei calcoli racchiusi nelle parentesi (.....)
2° - Elevamenti a potenza ↑
3° - Prodotti e divisioni * e /
4° - Addizioni e sottrazioni + e -

Per esempio, nell'istruzione:

```
10 LET X=A/B+C*D+E↑2
```

le operazioni vengono eseguite nel seguente ordine: in primo luogo si calcola l'elevamento a potenza E↑2; poi si esegue il prodotto C*D e la divisione A/B; infine si sommano i valori ottenuti da A/B, C*D, E↑2.

In base alla sequenza operativa dei calcoli ora indicata si deduce che l'espressione 2 riportata nella tabella della *fig. 6* è scritta correttamente; essa infatti si indica con:

```
Y=2*X+1
```

e non nel modo seguente:

```
Y=(2*X)+1.
```

In questo caso, infatti, le parentesi tonde sono di troppo, in quanto il sistema calcola prima 2*X e poi esegue la somma 2*X piú 1.

Nell'esempio 3), sempre della *fig. 6*, prima viene eseguita l'operazione racchiusa nella parentesi piú interna, che contiene SIN (X)+COS (Y); successivamente il risultato di questa somma viene addizionato a 1 e poi elevato al quadrato; a questo punto il sistema ha calcolato il risultato parziale dell'espressione:

$$[1+(\text{SIN } X+\text{COS } Y)]^2.$$

Quest'ultimo risultato viene poi sommato a 1 per ottenere il valore finale che verrà assegnato alla variabile di nome Z.

L'esempio 7) riporta una tipica espressione matematica: $\alpha=\beta+1$.

Come precisato in precedenza, espressioni letterali nella lingua greca antica non esistono in linguaggio BASIC, per cui al posto di α e β si sono utilizzati due nomi che suonano allo stesso modo: ALFA e BETA.

I nomi ALFA e BETA sono riconosciuti e interpretati dal computer come se fossero scritti AL e BE (i primi due caratteri di ALFA e BETA); in effetti, questo è un criterio formale di scrittura che può risultare utile a chi programma, tanto piú che il sistema non segnala errore.

Bisogna solo ricordare di non assegnare altri valori a nomi di variabili che cominciano per AL e BE, quali ad esempio:

ALFA, ALBA, ALTO;
BETA, BENA, BENE, ecc.

perché il computer li considererebbe tutti uguali, essendo appunto nomi di variabili che rappresentano gli stessi valori (AL e BE).

Per quanto riguarda gli operatori algebrici (+, –, *, /, ↑) vige la regola che questi **non possono** essere scritti in modo consecutivo, uno dopo l'altro; ad esempio, non è possibile scrivere:

```
A=+3*-2
```

bensí si deve scrivere:

```
A=+3*(-2)
```

mentre è possibile la scrittura:

```
A=3↑-3
```

per calcolare la potenza 3^{-3}.

Come si vede, le parentesi "aggiustano tutto", anche se un loro uso eccessivo provoca il rallentamento dell'elaborazione.

A proposito di parentesi, se in una istruzione il numero di quelle aperte risulta diverso dal numero di quelle chiuse, viene segnalato un messaggio di errore; prima di battere un'istruzione algebrica un po' lunga e complessa occorre quindi verificare che il numero delle parentesi aperte sia uguale al numero delle parentesi chiuse.

Occorre tenere presente, infine, che nel linguaggio BASIC **non sono ammesse le parentesi adiacenti** per indicare la moltiplicazione; ad esempio, (A+B) (C–D) è errato: si deve scrivere (A+B)*(C–D).

5. - Qualche dettaglio in piú sull'istruzione IF... THEN...

Relazioni semplici e multiple per le diramazioni condizionate

L'istruzione **IF... THEN...**, come si è visto nelle precendenti lezioni, consente al flusso dell'elaborazione di deviare in una certa altra parte di programma in base al valore (VERO o FALSO) di una relazione algebrica o proposizionale, relazione che è riportata dopo la parola riservata **IF**.

La relazione è un confronto che viene fatto tra costanti o variabili numeriche o alfanumeriche (stringhe) e può essere **semplice** o **multipla**.

Si ha una relazione semplice quando si mettono a confronto due entità omogenee, come nei seguenti casi:

```
1) 3<4
2) "A"<"B"
3) "BA">"AA"
4) X>=Y
5) X$<="A"
```

In alcune versioni BASIC l'istruzione condizionale **IF... THEN...** prevede anche la clausola **ELSE** (altrimenti), la quale specifica l'istruzione da eseguire nel programma se la condizione è falsa.

Per esempio:

```
10 IF A=0 THEN Y=B+2
   ELSE 30
```

che significa "se A=Ø viene allora eseguita l'assegnazione Y=B+2, altrimenti il programma passa all'istruzione 3Ø".

Una relazione è multipla quando invece è costituita da piú di una relazione semplice. Le relazioni multiple sono combinate secondo gli operatori logici (o booleani), i quali utilizzano operandi (variabili) che possono assumere soltanto due valori logici, convenzionalmente chiamati VERO e FALSO.

Nel linguaggio BASIC il valore VERO viene normalmente indicato con il

simbolo 1 oppure -1 (come nel caso dei computer C64 e VIC-20) ed il valore FALSO con il simbolo 0.

Gli operatori logici utilizzati nel linguaggio BASIC sono i seguenti.

- **AND** (prodotto logico di due o piú operandi): il risultato fornito è VERO solo nel caso in cui tutti gli operandi siano VERI; il risultato è FALSO se uno degli operandi (oppure tutti) è FALSO.
- **OR** (somma logica di due o piú operandi): il risultato fornito è VERO se almeno uno degli operandi è VERO; il risultato è FALSO se tutti gli operandi sono FALSI.
- **NOT** (negazione logica di un operando): il risultato è VERO quando l'operando è FALSO ed è FALSO quando l'operando è VERO.

Mediante gli operatori logici ora descritti, le relazioni (REL) algebriche o proposizionali possono essere cosí combinate:

```
REL 1    AND    REL 2
REL 1    OR     REL 2
         NOT    REL.
```

Per esempio, se si scrive l'istruzione:

```
10 IF A<B AND B<C THEN 500
20.....
```

il controllo dell'elaborazione passerà all'istruzione 5ØØ **se risultano** VERE **entrambe le relazioni**

```
A<B
B<C.
```

Se invece una delle due relazioni o entrambe risultano FALSE, il programma proseguirà per l'istruzione 2Ø...

Volendo fare un altro esempio, se si scrive:

```
10 IF A<B OR B<C THEN 500
20....
```

il controllo dell'elaborazione passerà all'istruzione 5ØØ **se è** VERA **"alme-**

no" una delle due relazioni

A<B
B<C,

altrimenti (cioè, se risultano entrambe FALSE) il controllo proseguirà per l'istruzione 2Ø.

L'operatore logico NOT REL indica l'opposto del valore (FALSO o VERO) della relazione REL, per cui l'istruzione:

IF NOT A<B **THEN** 5ØØ

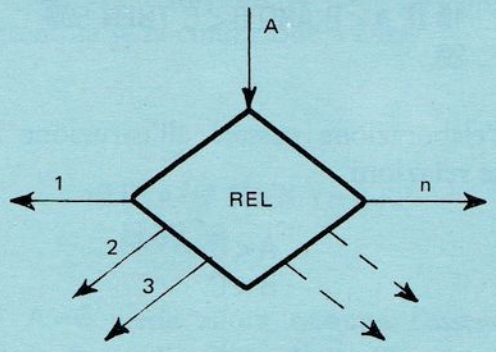
passerà il controllo dell'elaborazione a 5ØØ se A risulterà >=B.

Nella lezione successiva si studieranno le strutture di controllo che realizzano diramazioni semplici e multiple del flusso di elaborazione, diramazioni che si possono programmare anche con istruzioni **IF**.

Diramazioni condizionate multiple o nidificate

Analizziamo ora come è possibile effettuare una diramazione multipla con una serie di istruzioni **IF** (chiamate **IF** nidificate perché possono essere inserite una dentro l'altra).

Una diramazione multipla è cosí rappresentabile:



Il flusso dell'elaborazione che proviene da A può diramarsi in punti diversi del programma (1, 2, ...n) a seconda del valore di una variabile di controllo contenuta in una relazione algebrica o logica REL.

Un esempio di diramazione multipla (tripla) si ha quando occorre controllare dal programma se il valore di una variabile di nome X è maggiore, minore o uguale a Ø, in modo da far confluire il calcolo in punti diversi appropriati del programma.

Si supponga di voler controllare il valore di X nel calcolo dell'espressione:

$$A=\sqrt{\frac{1}{X}}\ .$$

Se X è minore di Ø, A diventa un numero immaginario:

$$A=\sqrt{\frac{1}{|X|}}\cdot i$$

dove | X | è il valore assoluto di X ed **i** è il **simbolo dei numeri immaginari, uguale per convenzione a** $\sqrt{-1}$.

Se X è uguale a Ø, $A=\infty$ (infinito).

Soltanto nel caso in cui X sia maggiore di Ø, A avrà un valore reale.

Il flow-chart relativo a tale esempio di diramazione tripla è riportato nella *fig. 7.*

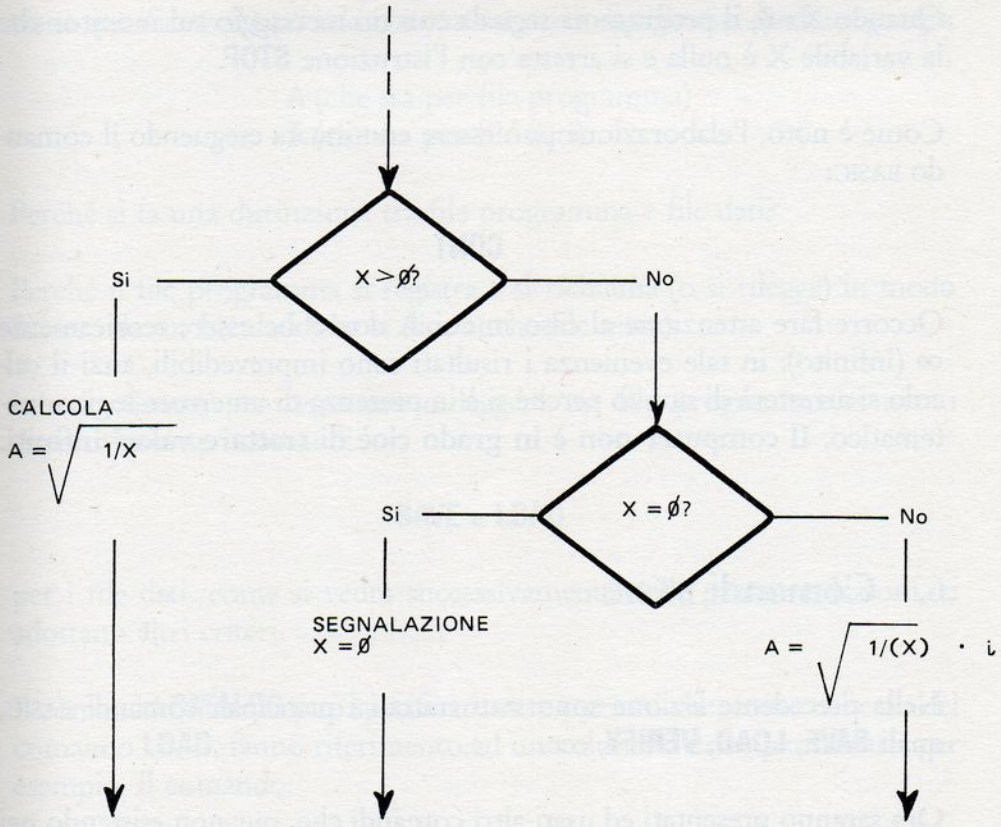


Fig. 7 – Flow-chart di una diramazione tripla.

Per codificare in linguaggio BASIC una diramazione tripla come quella della *fig. 7*, si possono utilizzare istruzioni **IF** nidificate nel modo riportato di seguito.

Esempio di diramazione multipla

```
100 IF X>0 THEN 130
105 IF X=0 THEN 140
110 REM*A E' UN NUMERO IMMAGINARIO*
115 REM*CON PARTE REALE DATA DA PR*
120 PR=SQR(1/ABS(X))
125 GOTO 150
130 A=SQR(1/X)
135 GOTO 150
140 REM*VIENE SRGNALATO CHE "X=0"*
145 PRINT "X=0"
150 STOP
155 ......
```

All'istruzione 120 si è usata la funzione matematica **ABS** (X), che consente di calcolare il valore assoluto di X (in pratica, se questo era negativo, con la funzione **ABS** diventa positivo).

Quando X=Ø, il programma segnala con un messaggio sul monitor che la variabile X è nulla e si arresta con l'istruzione **STOP**.

Come è noto, l'elaborazione può essere continuata eseguendo il comando BASIC:

CONT

Occorre fare attenzione al caso in cui A dovrebbe essere teoricamente ∞ (infinito): in tale evenienza i risultati sono imprevedibili, anzi il calcolo si arresterà di nuovo perché si è in presenza di un errore logico-matematico. **Il computer non è in grado cioè di trattare valori infiniti.**

6. – Comandi BASIC

Nella precedente lezione sono stati trattati i principali comandi BASIC, quali **SAVE**, **LOAD**, **VERIFY**, ecc.

Ora saranno presentati ed usati altri comandi che, pur non esistendo nei personal computer dotati di solo registratore a cassette, fanno ugual-

mente parte del linguaggio di programmazione BASIC; di conseguenza, nell'eventualità che si desiderasse accessoriare il personal computer con una unità a disco, questi comandi potranno rivelarsi particolarmente utili allo scopo.

Si inizierà con il comando BASIC:

CATALOG

il quale consente di visualizzare tutti i nomi dei file (archivi di programmi e dati) registrati su un floppy-disk, sempre che questo sia stato preventivamente inserito nel DRIVE (unità a disco).

In pratica, il comando **CATALOG** fa eseguire la lista di tutti i programmi e gruppi di dati registrati su un floppy-disk.

Di ogni programma sulla lista sono riportati il nome, il posto che occupa sul disco (espresso in numero di settori) e varie altre informazioni utili.

A seconda che si tratti di programma o di gruppo di dati, di ogni file viene specificato il tipo relativo con caratteri significativi quali i seguenti:

A (che sta per file programma)
T (che sta per file dati).

Perché si fa una distinzione tra file programma e file dati?

Perché il file programma si registra e si richiama (o si rilegge) in modo diverso da un file dati (che si dice anche file testo).

Infatti, mentre per registrare o leggere un file programma si usano rispettivamente i comandi:

SAVE e **LOAD**

per i file dati, come si vedrà successivamente nelle prossime lezioni, si adottano altri criteri.

Il comando **CATALOG** può specificare altre opzioni che, come nel caso del comando **LOAD**, fanno riferimento ad un volume o drive particolare; per esempio, il comando:

CATALOG VOL 1, D 2

esegue la lista dei file registrati sul floppy-disk di nome VOL 1, inserito nel DRIVE 2.

In alcuni altri sistemi personal computer, al posto del comando **CATALOG** viene utilizzato il comando **FILE**, seguito da altre opzioni che specificano nomi di volumi, file e drive.

Ad esempio, il comando:

FILES "Ø:VOL 5:/MY FILE"

cataloga le informazioni relative a file registrati su unità a disco (DRIVE Ø) su un volume di nome VOL 5 e con parola d'ordine MY FILE.

Praticamente questo comando esamina soltanto i file programmi che hanno una determinata parola d'ordine, la quale naturalmente non viene visualizzata; la parola d'ordine si comporta come un trucco di magia: c'è ma non si vede.

Si può anche facilmente comprendere il motivo di questa segretezza: la parola d'ordine, nota unicamente all'utente che l'ha programmata, serve a preservare dati e programmi da eventuali intromissioni o manomissioni di altri programmatori ed è molto usata quando si elaborano informazioni di una certa importanza, i cui risultati non si vogliono far conoscere ad altre persone.

Ulteriori, importanti comandi per la gestione dei file registrati su unità a disco sono: **DELETE**, **RENAME**, **LOCK**, **UNLOCK**, ecc.

Il comando **DELETE** (altri sistemi usano al suo posto il comando **SCRATCH**), seguito dal nome di un file, cancella quel file dal disco; esso serve per liberare settori di disco da programmi e dati non piú utilizzabili, riservando le zone liberate per altre informazioni.

Il comando **RENAME** (alcuni sistemi usano anche la parola **NAME**) consente di richiamare con un altro nome un file precedentemente registrato con un certo nome; per esempio il comando

RENAME PROG. 1, PROG. 2, D 2

assegna il nome PROG. 2 al file programma che precedentemente era stato denominato PROG. 1; questa operazione viene eseguita sull'unità a disco drive 2 (D2).

Altri sistemi utilizzano per la stessa operazione la seguente istruzione:

NAME "1:VOL 1:FILE 3" AS "1:VOL 1:FILE 4".

Con questo comando il file-programma chiamato FILE 3, contenuto nel volume disco di nome VOL 1 sul drive 1, viene "ridenominato" FILE 4.

I comandi **LOCK** e **UNLOCK** sono utilizzati per "bloccare" o "sbloccare" file registrati su disco, cioè per preservare gruppi di informazioni (programmi e dati) da eventuali operazioni di scrittura. Tali operazioni potrebbero essere effettuate da persone estranee o accidentalmente a seguito di errori da parte del sistema di elaborazione.

7. - Comandi di utilità

Oltre ai comandi descritti precedentemente, esiste, **almeno nei linguaggi BASIC di un certo livello**, un complesso di altri comandi, alcuni dei quali servono per la gestione automatica di una vasta gamma di unità periferiche, mentre altri sono utilizzati piú propriamente per agevolare il programmatore nel difficile compito di verifica della bontà di un'elaborazione piuttosto complessa.

Tra i primi si possono citare i comandi: **PR** # 1; **IN** # 1; **MON** C, I, O; ecc.

I comandi **PR** # 1 e **IN** # 1 consentono ad esempio di trasferire il controllo dell'elaborazione ad un terminale collegato, a cui è stato riservato il codice di periferica 1.

Piú esattamente, **PR** # 1 invia l'output dell'elaborazione alla periferica collegata allo SLOT di connessione n. 1 del computer, anziché al monitor (a cui è convenzionalmente assegnato lo SLOT Ø), mentre il comando **IN** # 1 preleva l'input dalla stessa periferica, anziché dalla tastiera dell'unità centrale.

Il comando **MON** C, I, O viene utilizzato per visualizzare (e quindi controllare) sul monitor la registrazione o la lettura di un flusso di dati su un'unità di memoria ausiliaria (a nastro o a disco): piú esattamente, **MON** C, I visualizza il flusso di dati immessi nella memoria principale dall'unità a disco, mentre **MON** C, O visualizza un flusso di dati proveniente dalla memoria principale e da registrare su una memoria a disco.

Un altro comando di utilità è il seguente:

MERGE

il quale consente di "collegare" un programma registrato su disco con uno presente nella memoria principale; naturalmente, **i due programmi devono avere i numeri di riferimento delle istruzioni non coincidenti,** altrimenti le istruzioni vanno a sovrapporsi le une sulle altre.

Il comando **MERGE** è utile per **allacciare varie subroutine** (sottoprogrammi) **ad un programma principale** presente in memoria, senza dovere ribattere i vari sottoprogrammi.

Tra i comandi utili per l'elaborazione e la raffigurazione grafica, vi sono le istruzioni:

TRACE e **NOTRACE**

Il comando **TRACE** (**NOTRACE** è la negazione di **TRACE**) consente di visualizzare ogni volta l'istruzione che sta per essere eseguita; ciò può risultare utile quando si vuole seguire "visivamente" un determinato calcolo algebrico.

L'istruzione **TRACE** può essere **ad esecuzione immediata o differita;** ogni volta che si vuole ritornare all'elaborazione normale, basta immettere l'istruzione **NOTRACE**.

L'istruzione **SPEED** (velocità) consente, invece, di variare la velocità di stampa o visualizzazione dei risultati.

Per esempio, se si scrive:

```
⋮
10 SPEED 128
20 PRINT "SCUOLA RADIO ELETTRA"
30 SPEED 256
40 PRINT "SCUOLA RADIO ELETTRA"
⋮
```

si vedranno apparire sullo schermo due volte le stringhe di uguale valore: "SCUOLA RADIO ELETTRA".

La prima stampa sarà effettuata con una velocità pari alla metà della

seconda (con una velocità 128 anziché 256, che è la massima consentita dal sistema).

Altre istruzioni che consentono un controllo "visivo" del programma sono quelle che effettuano delle pause o attese, quali le seguenti:

WAIT, PAUSE, ecc.

Tali attese possono naturalmente essere interrotte dall'operatore battendo un carattere qualsiasi da tastiera.

Praticamente l'utilizzo di queste pause condizionate permette una guida ragionata e pilotata dell'elaborazione.

8. – Istruzioni e comandi per la raffigurazione grafica

Come si è accennato nella lezione *Analisi e Programmazione* 2, il personal computer è particolarmente adatto alla **raffigurazione grafica automatica**, cioè alla progettazione programmata di figure geometriche.

Questa raffigurazione può avvenire su diversi supporti visivi, quali il monitor o il televisore, la stampante, il plotter, ecc.

Quando il sistema lo consente, la densità di raffigurazione può variare da una grandezza massima standard dei caratteri ad un'altra grandezza piú piccola: **è quella che si definisce grafica ad alta e bassa risolvenza.**

Nella grafica ad alta risolvenza i caratteri o punti di stampa (pixel) sono minutissimi, per cui possono essere addensati per coprire profili geometrici, mentre nella grafica a bassa risolvenza i caratteri sono raffigurati in grandezza normale standard e non consentono una risolvenza efficace dell'immagine.

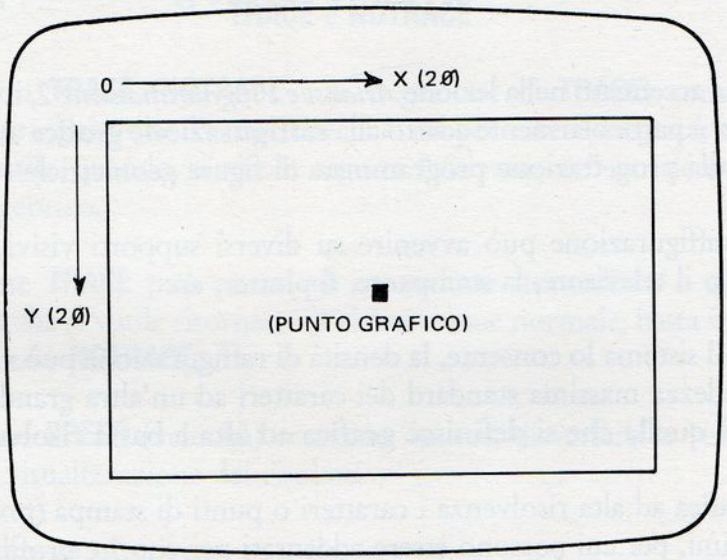
Tra le istruzioni BASIC che permettono di effettuare o "plottare" profili geometrici si possono citare le seguenti:

GR, HGR,
PLOT, HPLOT, TAB, HTAB, VTAB, HLINE, VLINE,
COLOR, HCOLOR,
POKE, PEEK, PDL,
SCALE, DRAW, XDRAW, ROT.

Queste istruzioni sono **valide per i sistemi** APPLE; per altri sistemi sono leggermente diverse nel senso che, pur avendo la stessa funzione grafica, sono denominate in modo differente oppure sono realizzate con modalità operative nettamente diverse, come ad esempio per quanto riguarda la rappresentazione dei colori, per la quale in alcuni microsistemi si utilizzano tasti speciali di colore.

Le istruzioni **GR** e **HGR** definiscono preliminarmente due diversi modi di rappresentazione grafica: a bassa risolvenza (**GR**) e ad alta risolvenza (**HGR**). Con **HGR** (APPLE II) è possibile raffigurare un profilo su una finestra costituita da 280×160 punti.

L'istruzione **PLOT** X, Y dispone un punto grafico a bassa risolvenza rispetto alle coordinate X, Y dello schermo. Per esempio, l'istruzione **PLOT** 2Ø, 2Ø fa apparire un punto grafico sullo schermo nel punto di ascissa X=2Ø e di ordinata Y=2Ø, come raffigurato nel disegnino che segue.



Come si può rilevare, lo schermo viene considerato come un sistema di assi cartesiani, con l'origine degli assi nell'angolo superiore sinistro di coordinate Ø,Ø, con l'asse X orientato verso destra e l'asse Y orientato verso il basso.

L'istruzione **HPLOT** consente di eseguire invece grafici ad alta risolvenza; per esempio:

HPLOT 1ØØ,1ØØ

raffigura un punto grafico minuscolo (pixel) su uno schermo piú denso, costituito da 280×160 punti.

L'istruzione **TAB** (X) porta il cursore nella posizione X a partire dal margine sinistro dello schermo e normalmente viene usata in un'istruzione **PRINT**.

Ad esempio:

PRINT TAB (5) "BASIC"

visualizzerà la parola "BASIC" dalla posizione 5 (oppure dalla colonna 6) a partire dal margine sinistro dello schermo, come si vede nella *fig. 8*.

Le istruzioni **HTAB** e **VTAB**, usate anch'esse in istruzioni **PRINT**, fanno muovere il cursore rispettivamente in senso orizzontale e in senso verticale.

HLINE e **VLINE** tracciano rispettivamente una linea orizzontale ed una linea verticale a partire da certe coordinate; ad esempio:

HLINE 5,3Ø AT 2Ø,1Ø

traccerà una linea orizzontale continua dal punto con coordinate 5,2Ø al punto con coordinate 3Ø,1Ø.

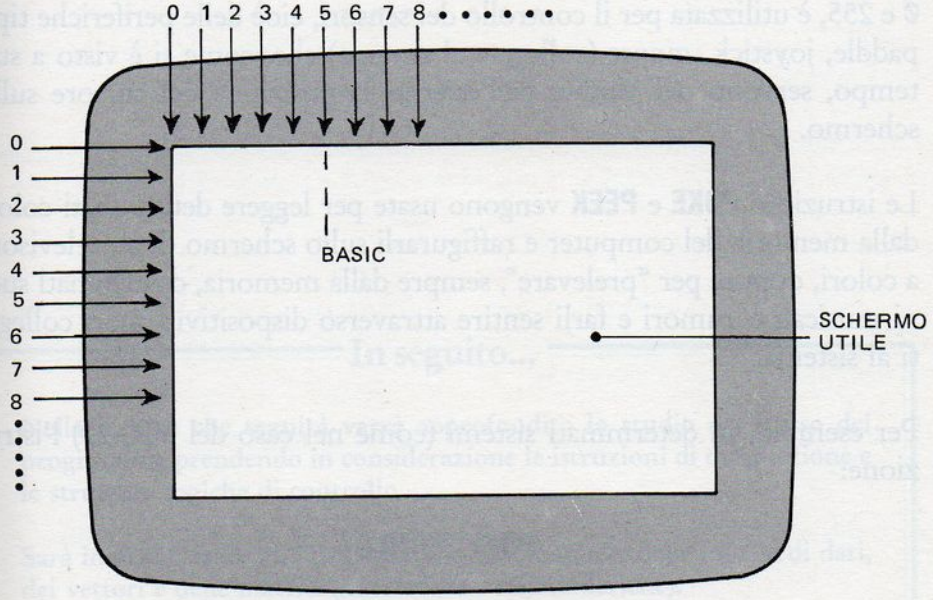


Fig. 8 – Esempio d'impiego dell'istruzione TAB (X).

Le istruzioni **COLOR** e **HCOLOR** consentono di raffigurare sullo schermo un certo numero di colori, che dipende dal particolare monitor a colori utilizzato dal personal computer.

Scrivendo:

COLOR=1

le successive visualizzazioni a bassa risolvenza saranno colorate con il colore n.1, che per certi sistemi è il codice rappresentativo del magenta.

Con l'istruzione:

HCOLOR=2

le successive visualizzazioni in alta risolvenza appariranno di colore blu, corrispondente appunto al codice di colore n. 2.

Generalmente, i colori raffigurabili sui monitor a colori dei personal computer sono sedici (con codici che vanno da Ø a 15).

Altri microsistemi utilizzano speciali tasti o istruzioni per la raffigurazione di una vasta gamma di combinazioni a colori.

L'istruzione **PDL**(X), dove X è un numero qualsiasi intero compreso tra Ø e 255, è utilizzata per il controllo dei sensori, cioè delle periferiche tipo paddle, joystick, mouse (collegate al sistema) che, come si è visto a suo tempo, servono per variare dall'esterno la posizione del cursore sullo schermo.

Le istruzioni **POKE** e **PEEK** vengono usate per leggere determinati colori dalla memoria del computer e raffigurarli sullo schermo di un televisore a colori, oppure per "prelevare", sempre dalla memoria, determinati suoni musicali o rumori e farli sentire attraverso dispositivi sonori collegati al sistema.

Per esempio, in determinati sistemi (come nel caso del VIC-20) l'istruzione:

POKE 36879,8

permette di visualizzare sullo schermo a colori il colore di codice 8 (nero).

Le istruzioni **POKE** e **PEEK** non vengono però utilizzate solamente per inserire o prelevare colori o suoni dalla memoria del computer; esse vengono impiegate, come avrà modo di constatare in una prossima lezione, per inserire o prelevare dalla memoria del computer dei valori particolari, che possono servire ad alterare le caratteristiche di funzionamento standard del computer stesso.

Le istruzioni **POKE** e **PEEK** in questione possono altresí essere impiegate per introdurre istruzioni in linguaggio macchina.

Infine, le istruzioni **DRAW**, **XDRAW**, **ROT**, **SCALE**, utilizzate nella realizzazione dei profili grafici ad alta risolvenza, consentono alla punta grafica di tracciamento di subire spostamenti infinitesimi per tutti i punti (pixel) dello schermo predisposto per l'alta risolvenza. Tali spostamenti possono essere orientati in tutte le direzioni.

* * * * * * *

In seguito...

Nella lezione che seguirà verrà approfondito lo studio del flusso del programma, prendendo in considerazione le istruzioni di diramazione e le strutture logiche di controllo.

Sarà inoltre esaminata l'utilizzazione sul computer delle tabelle di dati, dei vettori e delle matrici (numeriche e alfanumeriche).

Da memorizzare

- In linguaggio BASIC per assegnare una etichetta o un nome ad un valore si usa l'istruzione **LET**, oppure si scrive una semplice equazione che funge da assegnazione.
- I nomi delle variabili reali si differenziano dai nomi delle variabili intere per il fatto che queste ultime riportano dopo i due caratteri alfanumerici il simbolo %, mentre le variabili reali non hanno tale carattere.
- I valori assegnati alle variabili reali sono anche detti **a virgola fissa**, in quanto la posizione del punto decimale non varia quando il numero deve essere memorizzato o visualizzato; ciò vale se il numero è compreso nell'intervallo: | Ø . Ø1 | ÷ | 999999999.9 |. I valori numerici reali che sono fuori dal suddetto intervallo vengono trasformati in **notazione esponenziale.**
- La notazione esponenziale consiste nel trasformare il numero nel prodotto di una mantissa (parte piú significativa del numero) per una potenza di 1Ø, in modo che il valore resti invariato.
- Nel linguaggio BASIC la notazione esponenziale viene attuata sostituendo il numero 1Ø (base della potenza) con la lettera E, la quale indica appunto che il numero è rappresentato in notazione esponenziale.
- I numeri interi nel linguaggio BASIC conservano la propria forma intera se il loro valore è compreso nell'intervallo: -32768 ÷ +32767.
- La stringa è **un insieme di lettere, numeri e caratteri grafici**, delimitati all'inizio ed alla fine dal carattere virgolette ("). Il nome usato per indicare le variabili stringa termina sempre con il carattere $.
- Le variabili numeriche (reali ed intere) e le variabili stringa occupano nella memoria principale 7 byte.
- Nella memoria del computer tutti i caratteri riportati dopo la parola chiave occupano 1 byte ciascuno (ve ne possono essere al massimo 239).
- La sintassi del BASIC prevede il seguente ordine di precedenza per l'esecuzione delle operazioni algebriche: risoluzione delle parentesi; elevamento a potenza; prodotti e divisioni; addizioni e sottrazioni.
- L'istruzione condizionale **IF... THEN...** consente al flusso di elaborazione di deviare per una certa altra parte di programma, in base al valore VERO o FALSO di una relazione algebrica o proposizionale.
- Gli operatori logici utilizzati nel linguaggio BASIC sono: AND, OR e NOT.
- L'operatore AND fornisce il risultato VERO soltanto nel caso in cui tutti gli operandi siano VERI; l'operatore OR fornisce il risultato VERO se almeno uno degli operandi è VERO; l'operatore NOT fornisce il risultato VERO quando l'operando è FALSO e viceversa.
- I principali comandi per la gestione dei file (archivi di programmi e dati) sulle unità a disco sono: **CATALOG**, **DELETE**, **RENAME**, **LOCK**, **UNLOCK**, ecc.
- I comandi piú comuni per la gestione automatica delle unità periferiche sono: **PR** # 1; **IN** # 1; **MON** C, I, O; ecc.
- Per la raffigurazione grafica vengono usati appositi comandi ed istruzioni (come per esempio: **GR**, **PLOT**, **TAB**, **COLOR**, **POKE**, **PEEK**, **SCALE**, ecc.), che differiscono da un tipo di computer all'altro.

Scuola Radio Elettra
Torino

Pratica del BASIC

Lezione 5

1 — Ordinamento dei dati da stampare
2 — Costanti e variabili
3 — Le espressioni algebriche
4 — Esercizi di programmazione
5 — Consigli pratici per la programmazione del computer

Che cosa farà

In questa lezione avrà modo di analizzare il comportamento delle funzioni **TAB** (X) e **SPC** (X), utilizzate nell'interno della **PRINT**, le quali consentono rispettivamente di stampare in colonna dei dati e di spaziare fra loro due dati.

Verranno poi presi in considerazione le rappresentazioni delle costanti e delle variabili nel computer e l'impiego di quest'ultimo nel calcolo delle espressioni algebriche.

Negli esercizi di programmazione previsti eseguirà, fra l'altro, un programma di calcolo, nel quale sono contemplate diverse istruzioni di diramazione condizionate **IF**, con relazioni logiche di tipo AND.

Infine, verranno forniti utili consigli per la stesura dei programmi.

1. - Ordinamento dei dati da stampare

Nella precedente lezione pratica del BASIC ha potuto constatare che, quando i dati (variabili o costanti) nell'interno dell'istruzione **PRINT** sono separati da una virgola, vengono riprodotti sullo schermo secondo una suddivisione standard della riga di stampa.

In altre parole, la riga di stampa viene automaticamente divisa in quattro zone, ognuna delle quali costituita da dieci colonne che condizionano l'incolonnamento dei dati.

Per ottenere una diversa spaziatura dei dati sullo schermo, si può inserire nell'interno della **PRINT** la funzione **TAB** (X) oppure la funzione **SPC** (X).

La funzione **TAB** (X), denominata **tabulatore**, sposta il cursore in corrispondenza della colonna numero X: tenga presente che le colonne nel Suo computer sono numerate da 0 a 39.

Vediamo ora come lavora la funzione **TAB**. Accenda il microcomputer ed il televisore ad esso collegato; batta sulla tastiera il semplice programma che segue, ricordando di premere sempre il tasto **RETURN** al termine di ogni linea.

```
10 PRINT TAB (5);"B"
20 PRINT TAB (8);"A"
30 PRINT TAB (11);"S"
40 PRINT TAB (14);"I"
50 PRINT TAB (17);"C"
```

Esegua quindi il programma, scrivendo il comando **RUN** seguito dal tasto **RETURN**.

Sullo schermo apparirà:

```
   B
      A
         S
            I
               C
```

Ossia la lettera B risulterà stampata a partire dalla colonna 5, la lettera A a partire dalla colonna 8, la lettera S a partire dalla colonna 11 e cosí via.

Come ha potuto verificare da questo esempio, la funzione **TAB** specifica la colonna in cui deve apparire il carattere successivo.

Tale istruzione viene comunemente impiegata per allineare le stampe dei dati in colonne verticali, in corrispondenza di un prefissato punto di tabulazione.

La funzione **SPC** (X), anch'essa usata solamente nell'interno dell'istruzione **PRINT**, a differenza della **TAB**, consente di spostare il cursore di un certo numero di colonne anziché portarlo ad una colonna prestabilita.

In altri termini, la funzione **SPC** (X) permette di spaziare fra loro due dati di una certa quantità di spazi X.

Per verificare quanto asserito, cancelli dalla memoria del computer il precedente programma, scrivendo il comando BASIC **NEW** seguito dal tasto **RETURN**.

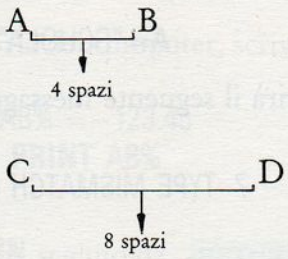
Pulisca poi lo schermo premendo il tasto **SHIFT** e battendo contemporaneamente il tasto **CLR | HOME**.

Scriva ora le seguenti due istruzioni differite, premendo il tasto **RETURN** al termine di ognuna di esse:

```
10 PRINT "A"; SPC (4);"B"
20 PRINT "C"; SPC (8);"D"
```

Mandi poi in esecuzione quanto ha battuto, scrivendo il comando **RUN** e premendo il tasto **RETURN**.

Sullo schermo vedrà apparire:



Le lettere A e B risulteranno separate tra loro da 4 spazi, mentre le lettere C e D appariranno separate da 8 spazi, come appunto indicato dallo specificatore SPC.

Cancelli con il solito sistema la memoria del computer e lo schermo.

2. - Costanti e variabili

Nei microsistemi i nomi assegnati a costanti e variabili numeriche o stringa **sono costituiti al massimo da due caratteri alfanumerici** (lettere e cifre decimali); ogni nome però **inizia sempre con una lettera** e, a seconda che si tratti di valore intero o stringa, esso è seguito, dopo i primi due caratteri, rispettivamente dai caratteri speciali % e $.

Per quanto riguarda i campi di validità di numeri e stringhe, essi sono specificati nella tabella che segue.

TIPO DI COSTANTI O VARIABILI	CAMPO DI VALIDITA'
Costante o variabile intera	$-32768/+32767$
Costante o variabile reale	$+1{\cdot}10^{-38}/+1{\cdot}10^{+38}$
Costante o variabile stringa	80 caratteri alfanumerici per C64 (due righe di stampa) 160 caratteri alfanumerici per C128 (quattro righe di stampa)

Vediamo ora di verificare praticamente quanto asserito.

Scriva l'istruzione immediata che segue e le successive, premendo il tasto **RETURN** al termine di ognuna di esse:

```
A="SCUOLA"
```

Sullo schermo apparirà il seguente messaggio d'errore:

```
? TYPE MISMATCH ERROR
```

perché alla stringa di caratteri "SCUOLA" è stato erroneamente assegnato il nome A anziché A$.

Lo stesso messaggio d'errore verrà fornito, come Lei stesso potrà verificare, scrivendo le seguenti istruzioni di assegnazione errate:

```
a)  B$=SCUOLA
b)  B%="SCUOLA"
c)  A$=890
```

Nell'istruzione a) la parola SCUOLA è stata scritta senza virgolette, per cui non viene riconosciuta dal computer come stringa di caratteri; nel caso b) è stata assegnata una stringa di caratteri alla costante intera B%, che prevede dati numerici; nel caso c), invece, è stato assegnato un dato numerico alla costante stringa di nome A$.

Scriva ora quanto segue, premendo **RETURN** al termine di ogni riga:

```
10 AB%=123.45
15 PRINT AB%
```

Batta poi il comando BASIC **RUN** e prema **RETURN**.

Sullo schermo apparirà:

```
123
```

ossia il numero 123.45 verrà riprodotto senza la parte decimale (.45) ed assegnato alla variabile AB%.

Resta così confermato che, quando il nome di una variabile termina con il suffisso %, essa accetta soltanto numeri interi (compresi fra –32768 e +32767), troncando gli eventuali decimali.

Questo tipo di variabile viene utilizzata, quando è possibile, per risparmiare spazio nella memoria del computer.

Dopo aver cancellato la memoria del computer, scriva:

```
10AB% = -123.45
15 PRINT AB%
```

Batta poi il comando BASIC **RUN** seguito da **RETURN** .

Sullo schermo apparirà:

```
-124
```

In questo caso, il numero essendo negativo, viene anche arrotondato.

Per la rappresentazione dei valori numerici, come si è visto nella lezione *Analisi e Programmazione 5*, si utilizzano due tipi di notazione, e precisamente i seguenti.

1) **Rappresentazione in virgola fissa** per i valori reali compresi fra i limiti:

   | 0.01 | ÷ | 99999999.9 |

   dove con | numero | viene indicato il valore assoluto del numero stesso.

   Per esempio i numeri:

   -3418.64 ; +0.02 ; -15.01 ; +4020

   vengono tutti rappresentati in virgola fissa, ossia cosí come sono scritti.

2) **Rappresentazione esponenziale o scientifica** (detta anche **a virgola mobile** o **floating point**), per valori numerici reali al di fuori dell'intervallo citato prima.

   Per esempio i numeri:

   1234567809.01 ; $3.4*10^{10}$; 0.0000000001

vengono rispettivamente rappresentati nel seguente modo:

1.23456781E+Ø9 ; 3.4ØØØØØØ1E+1Ø ; 1E-1Ø.

Come si può osservare, la notazione esponenziale è costituita da una prima cifra significativa e da un massimo di otto cifre decimali seguite dalla lettera E (la quale significa 10 elevato a...), a sua volta seguita dall'esponente a cui viene elevato 10, preceduto da + oppure - per riavere il numero dato.

Le cifre oltre la nona significativa vengono eliminate ed eventualmente provocano un arrotondamento all'ultima cifra rappresentata; ad esempio il valore:

$$-1{,}345277892 \times 10^{12}$$

viene cosí rappresentato:

-1.34527789E+12.

Cancelli ora lo schermo con il solito sistema e scriva le istruzioni ad esecuzione immediata riportate di seguito, ricordando di premere il tasto **RETURN** al termine di ciascuna di esse.

Accanto a ciascuna di queste istruzioni è riportato il corrispondente valore visualizzato sullo schermo.

	Istruzioni	**Valori visualizzati**
a)	**PRINT** -Ø.Ø9	-.Ø9
b)	**PRINT** Ø.ØØ1	1E-Ø3
c)	**PRINT** 11.34566744	11.3456675
Computer C64 → d)	**PRINT** "AA...(85 volte A)"	? SINTAX ERROR
Computer C128 → e)	**PRINT** "AA...(165 volte A)"	? STRING TOO LONG ERROR
f)	**PRINT** 12345678.9	12345678.9
g)	**PRINT** 2.34567Ø9E+4Ø	? OVERFLOW ERROR
h)	**PRINT** 2.34567Ø9E-4Ø	Ø
i)	**PRINT** 99999999999	1E+11

Nel caso a) si può osservare che la parte intera, essendo nulla (Ø), viene omessa; nel caso d) compare il messaggio d'errore, in quanto la stringa risulta di lunghezza superiore a 80 caratteri; nel caso e) compare il mes-

saggio d'errore in quanto la stringa ha più di 160 caratteri; nel caso g) compare il segnale d'errore che indica il supero di capacità, in quanto il numero assegnato è maggiore di quello massimo ($1 \cdot 10^{+38}$) ammesso dal sistema; nel caso h), essendo il numero scritto troppo piccolo, il computer arrotonda tale numero a zero (Ø).

3. - Le espressioni algebriche

Nel calcolatore, come già precisato, le espressioni algebriche sono svolte nel seguente ordine.

Dapprima vengono risolte le operazioni **contenute fra parentesi**, a partire da quelle più interne; poi si procede con **gli eventuali elevamenti a potenza**, quindi con **le moltiplicazioni** e **le divisioni**; infine si eseguono **le sottrazioni** e **le addizioni**. Tra le moltiplicazioni e le divisioni viene eseguito prima il tipo di operazione scritto più a sinistra. L'ordine, quindi, è esattamente eguale a quello previsto dall'algebra.

Nel caso si debbano calcolare espressioni molto lunghe e complesse, è opportuno spezzarle in più parti e poi combinare queste ultime per ottenere il risultato finale.

Simile criterio è opportuno che venga quasi sempre usato nel Suo microcomputer.

Ricordi che l'elevamento a potenza viene ottenuto premendo il tasto ↑ , il quale riporta sul lato anteriore il simbolo π.

Supponiamo ora, per esempio, di volere calcolare il risultato della seguente espressione algebrica:

$$Y = \underbrace{\frac{1+\left[2+\left(3+\frac{1}{2}\right)^{2}\right]\cdot\frac{18}{20}}{\frac{1}{2}+\frac{3}{4}\left(\frac{70+5}{22}\right)}}_{Y_1} + \underbrace{\frac{\left(\frac{7}{8}+\frac{20}{22}\right)^{-\frac{1}{2}}+\left(\frac{17}{20}\right)^{2}}{\frac{6}{8}+\frac{2}{3}\left(3\right)^{-\frac{1}{2}}}}_{Y_2} + \underbrace{\left[1+\left(\frac{13}{17}+\frac{16}{20}\right)^{3}\right]}_{Y_3}$$

Per maggiore semplicità conviene calcolare separatamente i risultati delle tre espressioni Y_1, Y_2, Y_3, quindi sommare tra loro i valori di queste ultime per ottenere il risultato Y.

A titolo di esercizio, esegua perciò le tre espressioni indicate prima, scrivendo le istruzioni immediate che seguono e premendo **RETURN** al termine di ciascuna di esse:

```
Y1=(1+(2+(3+1/2)↑2)*18/20)/(1/2+3/4*((70+5)/22))
Y2=((7/8+20/22)↑-0.5+(17/20)↑2)/(6/8+2/3*(3)↑-0.5)
Y3=(1+(13/17+16/20)↑3)
Y=Y1+Y2+Y3
PRINT Y
```

Otterrà sullo schermo:

```
10.6498537
```

che è il risultato dell'espressione algebrica di partenza (Y).

Ricordi, come regola generale, che **in ogni espressione aritmetica il numero delle parentesi rotonde aperte deve risultare sempre uguale al numero delle parentesi rotonde chiuse**; in caso contrario apparirà sullo schermo il messaggio:

```
? SYNTAX ERROR
```

E' consigliabile, soprattutto nel primo periodo di apprendimento del linguaggio BASIC, "abbondare" nell'uso di parentesi anche se ciò comporta un maggiore onere per il computer.

Noti che nelle espressioni Y_1 e Y_3 dell'esempio fatto, al posto delle parentesi quadre, non previste nel BASIC, sono state usate parentesi tonde.

I simboli delle operazioni aritmetiche riportati nella lezione *Analisi e Programmazione 5* valgono anche per il Suo microsistema.

Per quanto riguarda il calcolo delle potenze, gli esponenti di queste ultime possono essere positivi o negativi; ad esempio, è corretta l'indicazione:

```
10↑-2,
```

che può anche essere scritta:

```
1/10↑2
```

in quanto dalla matematica sappiamo che:

$$10^{-2} = \frac{1}{10^2}.$$

L'esponente di una potenza può essere anche frazionario; per esempio:

$$2^{-\frac{1}{2}}$$

che in BASIC si scrive:

```
2↑-0.5
```

corrispondente in matematica al valore:

$$\frac{1}{\sqrt{2}} = 0.707106782.$$

Esegua ora la seguente istruzione immediata, che prevede il calcolo di 4^2 e 5^3:

```
PRINT 4↑2, 5↑3
```

e poi prema **RETURN** .

Sullo schermo compariranno, come previsto, i valori:

```
 16          125
```

che sono appunto i risultati di 4^2 e 5^3.

Scriva ora, premendo sempre il tasto **RETURN** alla fine, l'istruzione:

```
PRINT 7↑2.
```

Vedrà apparire sullo schermo il numero:

```
49.0000001
```

anziché 49, come dovrebbe risultare calcolando 7^2.

Anche se 49.ØØØØØØ1 è una buona approssimazione di 49, non è sicuramente un risultato esatto.

Nella maggior parte dei calcoli questa imprecisione ha però scarse conseguenze.

Certamente Lei si chiederà come mai l'imprecisione del computer si è manifestata calcolando 7^2, mentre negli altri due calcoli, 4^2 e 5^3, il risultato fornito era esatto.

La ragione di questo strano comportamento va attribuita al fatto che nel computer l'operazione di elevazione di un numero ad una data potenza non viene eseguita moltiplicando il numero per sé stesso tante volte quanto è il valore dell'esponente, ma utilizzando i logaritmi, i quali forniscono risultati piú o meno approssimati.

Di conseguenza, l'elevamento a potenza di un numero può risultare anch'esso approssimato.

Nel computer vengono utilizzati i logaritmi perché consentono di aumentare sensibilmente la velocità di calcolo.

Il computer permette anche il calcolo diretto della radice quadrata

$$\sqrt{X}$$

la quale viene cosí codificata in linguaggio BASIC:

SQR (X)

(le lettere SQR ricordano le parole inglesi "square root", cioè radice quadrata)

dove X è un numero positivo oppure Ø.

Per esempio, batta sulla tastiera la seguente istruzione immediata:

PRINT SQR (16)

e poi prema **RETURN** .

Sullo schermo comparirà il numero:

4

che è appunto il risultato di $\sqrt{16}$.

Provi ora a battere:

PRINT SQR (-25)

seguito da **RETURN**.

Sullo schermo comparirà il seguente messaggio d'errore:

? ILLEGAL QUANTITY ERROR

perché il numero, essendo negativo, è fuori dal campo ammesso.

Nelle espressioni algebriche possono anche essere usati gli **operatori relazionali**, ai quali viene sostituito il valore –1 se la relazione è verificata ed il valore Ø se essa non è verificata.

Ad esempio, scriva:

PRINT 45*(9<4)

quindi prema **RETURN**. Otterrà come risultato:

Ø

infatti, non essendo vera la relazione 9<4, ad essa viene sostituito il valore Ø.

Scriva ora:

PRINT 45*(9>4)

sempre seguito da **RETURN**. Otterrà:

-45

perché la relazione 9>4 è vera e quindi ad essa viene sostituito il valore –1, per cui 45*(–1)=–45.

4. - Esercizi di programmazione

1° ESERCIZIO

Dalla geometria sappiamo che l'equazione di una retta posta su un piano e rappresentabile su un sistema di assi cartesiani ortogonali si scrive nel seguente modo:

$$1)\ aX+bY+c=0$$

dove a, b, c sono parametri numerici che identificano la posizione relativa della retta rispetto all'origine degli assi cartesiani e l'inclinazione della stessa rispetto all'asse X, mentre X, Y sono due variabili reali, la prima indipendente e la seconda dipendente dalla prima: per ogni valore indipendente (qualunque esso sia) di X si ricava cioè un solo valore di Y.

Se si traccia il grafico dei punti X, Y su un piano di riferimento cartesiano, si ottiene la figura geometrica di una retta, la cui equazione è appunto data dall'espressione 1).

Se si scrive l'espressione 1) in funzione di Y, si ottiene:

$$2)\ Y= -\frac{a}{b}X -\frac{c}{b}$$

che è equivalente all'equazione 1), però la variabile Y assume nell'ultimo caso il ruolo di funzione lineare dipendente, mentre X assume il ruolo di variabile indipendente.

L'equazione 2) rappresenta una generica retta del piano; se si assegnano determinati valori numerici ai parametri a, b, c, la retta da generica diventa particolare (cioè una sola tra le infinite rette del piano).

Per esempio, se si assegnano i valori:

$$a=2$$
$$b=-3$$
$$c=+3$$

si ottiene la retta particolare di equazione (*fig. 1*):

$$Y=\frac{2}{3}X+1$$

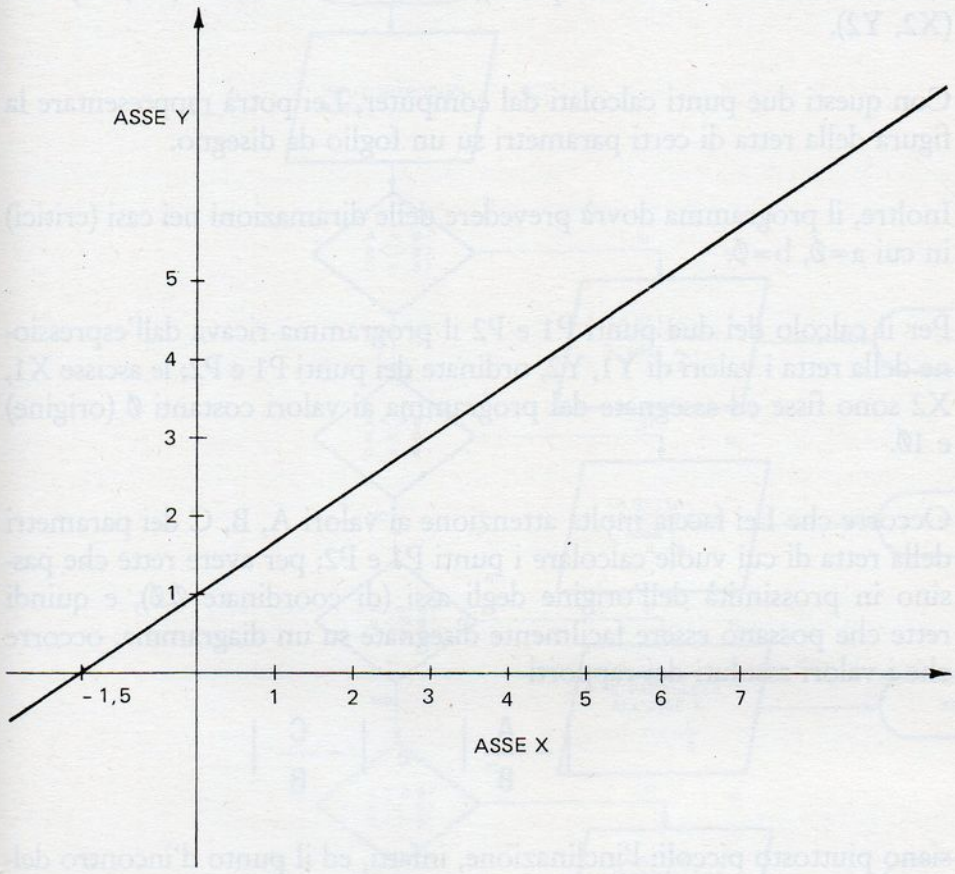


Fig. 1 – Rappresentazione grafica della retta di equazione $Y = \frac{2}{3}X + 1$.

Per ogni combinazione di parametri a, b, c, si ottengono rette diverse: rette che passano per l'origine degli assi cartesiani (caso in cui c=0); rette inclinate di un certo angolo (la cui tangente è data dal valore –a/b); rette parallele all'asse X (caso in cui a=0); rette parallele all'asse Y (caso in cui b=0).

Fatta questa premessa, Lei può creare un programma BASIC che Le consenta di ottenere la raffigurazione di qualsiasi retta del piano, dati i valori dei tre parametri a, b, c.

Questi ultimi, che potrà chiamare A, B, C, dovranno essere introdotti da tastiera.

Il programma dovrà calcolare almeno due punti della retta (cioè le coordinate cartesiane X, Y di due punti), che indicherà con P1 (X1, Y1) e P2 (X2, Y2).

Con questi due punti calcolati dal computer, Lei potrà rappresentare la figura della retta di certi parametri su un foglio da disegno.

Inoltre, il programma dovrà prevedere delle diramazioni nei casi (critici) in cui a=Ø, b=Ø.

Per il calcolo dei due punti P1 e P2 il programma ricava dall'espressione della retta i valori di Y1, Y2, ordinate dei punti P1 e P2; le ascisse X1, X2 sono fisse ed assegnate dal programma ai valori costanti Ø (origine) e 1Ø.

Occorre che Lei faccia molta attenzione ai valori A, B, C dei parametri della retta di cui vuole calcolare i punti P1 e P2; per avere rette che passino in prossimità dell'origine degli assi (di coordinate Ø,Ø), e quindi rette che possano essere facilmente disegnate su un diagramma, occorre che i valori assoluti dei rapporti

$$\left| -\frac{A}{B} \right| \quad \text{e} \quad \left| -\frac{C}{B} \right|$$

siano piuttosto piccoli: l'inclinazione, infatti, ed il punto d'incontro della retta (con parametri A, B, C) con l'asse Y sono dati dai rapporti:

$-\frac{A}{B}$ e $-\frac{C}{B}$ che devono essere appunto bassi.

Nella *fig. 2* è rappresentato il flow-chart del programma in questione, che verrà denominato RETTA DEL PIANO.

Come si può rilevare, in questo diagramma sono presenti sei istruzioni di salto condizionato, una per ciascuna delle combinazioni dei parametri A, B, C.

Nel blocco 1) è rappresentata un'istruzione di **INPUT** con visualizzazione del messaggio

"INTRODUCI A, B, C"

il quale avverte l'operatore che deve introdurre i tre valori numerici per calcolare i due punti della retta.

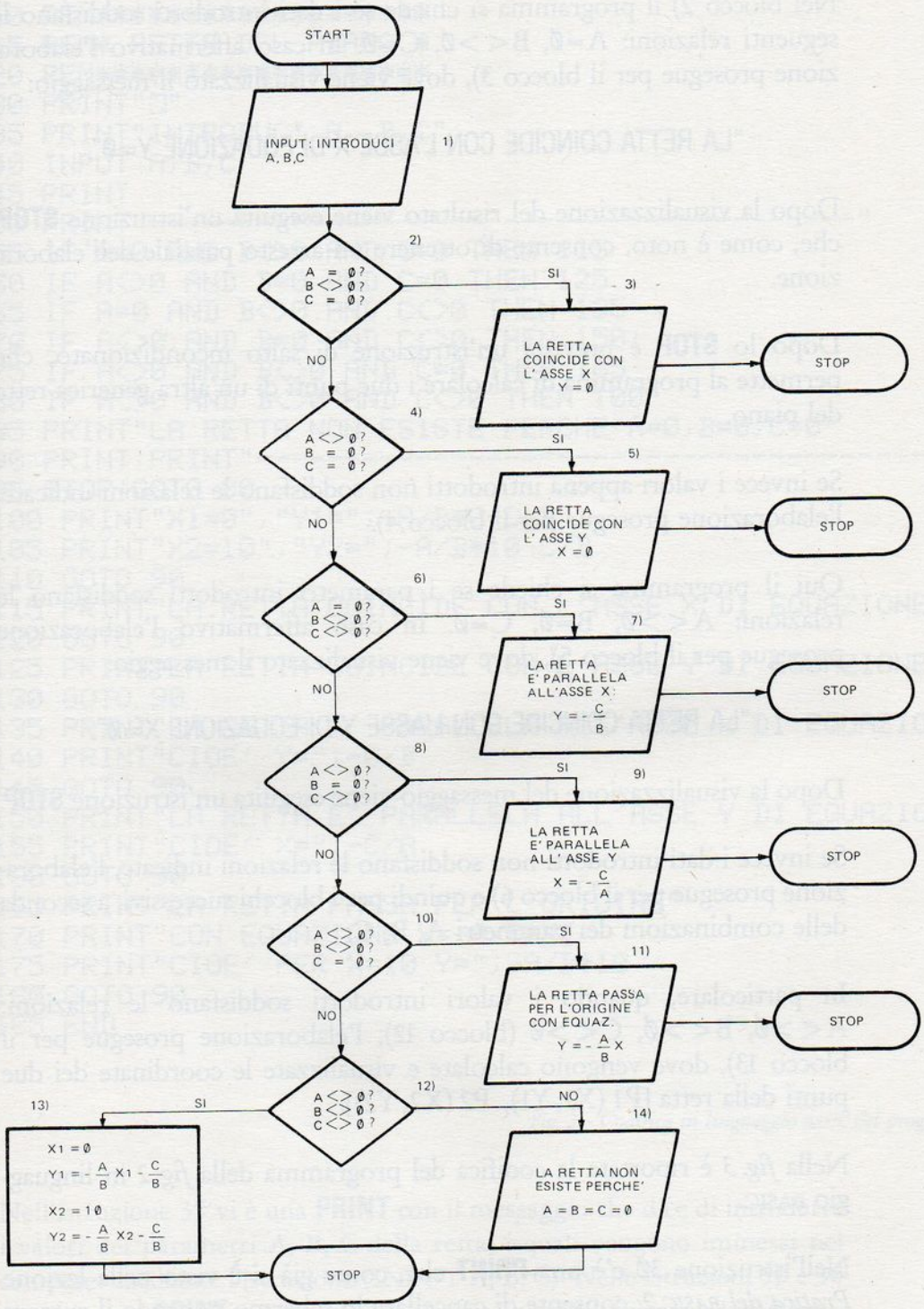


Fig. 2 – Flow-chart del programma che calcola due punti, P1 e P2, di una retta del piano di equazione $Y = -\frac{a}{b}X - \frac{c}{b}$. *Le ordinate* Y_1, Y_2 *dei punti P1 e P2 sono calcolate per i valori d'ascisse* $X_1 = Ø$, $X_2 = 1Ø$.

Nel blocco 2) il programma si chiede se i dati introdotti soddisfano le seguenti relazioni: A=Ø, B<>Ø, C=Ø. In caso affermativo l'elaborazione prosegue per il blocco 3), dove viene visualizzato il messaggio:

"LA RETTA COINCIDE CON L'ASSE X DI EQUAZIONE Y=Ø".

Dopo la visualizzazione del risultato viene eseguita un'istruzione **STOP**, che, come è noto, consente di ottenere un arresto parziale dell'elaborazione.

Dopo lo **STOP** è prevista un'istruzione di salto incondizionato, che permette al programma di calcolare i due punti di un'altra generica retta del piano.

Se invece i valori appena introdotti non soddisfano le relazioni indicate, l'elaborazione prosegue per il blocco 4).

Qui il programma si chiede se i parametri introdotti soddisfano le relazioni: A<>Ø, B=Ø, C=Ø. In caso affermativo l'elaborazione prosegue per il blocco 5), dove viene visualizzato il messaggio:

"LA RETTA COINCIDE CON L'ASSE Y DI EQUAZIONE X=Ø".

Dopo la visualizzazione del messaggio viene eseguita un'istruzione **STOP**.

Se invece i dati introdotti non soddisfano le relazioni indicate, l'elaborazione prosegue per il blocco 6) e quindi per i blocchi successivi, a seconda delle combinazioni dei parametri A, B, C.

In particolare, quando i valori introdotti soddisfano le relazioni: A<>Ø, B<>Ø, C<>Ø (blocco 12), l'elaborazione prosegue per il blocco 13), dove vengono calcolate e visualizzate le coordinate dei due punti della retta [P1 (X1, Y1), P2 (X2, Y2)].

Nella *fig. 3* è riportata la codifica del programma della *fig. 2* in linguaggio BASIC.

Nell'istruzione 3Ø c'è una **PRINT** che, come già si è visto nella lezione *Pratica del BASIC 2*, consente di cancellare lo schermo ponendo il cursore sull'angolo sinistro dello stesso. Questa **PRINT** viene eseguita premendo il tasto **SHIFT** e battendo contemporaneamente i tasti:

" | 2 → **CLR | HOME** → **" | 2** .

```
10 REM********************
15 REM* RETTA DEL PIANO *
20 REM********************
30 PRINT"□"
35 PRINT"INTRODUCI A, B,C"
40 INPUT A,B,C
45 PRINT
50 PRINT"----------------------------------------"
55 IF A=0 AND B<>0 AND C=0 THEN 115
60 IF A<>0 AND B=0 AND C=0 THEN 125
65 IF A=0 AND B<>0 AND C<>0 THEN 135
70 IF A<>0 AND B=0 AND C<>0 THEN 150
75 IF A<>0 AND B<>0 AND C=0 THEN 165
80 IF A<>0 AND B<>0 AND C<>0 THEN 100
85 PRINT"LA RETTA NON ESISTE PERCHE'A=0,B=0,C=0"
90 PRINT:PRINT"----------------------------------------"
95 STOP:GOTO 30
100 PRINT"X1=0","Y1=";-A/B*0-C/B
105 PRINT"X2=10","Y2=";-A/B*10-C/B
110 GOTO 90
115 PRINT"LA RETTA COINCIDE CON L'ASSE X DI EQUAZIONE Y=0"
120 GOTO 90
125 PRINT"LA RETTA COINCIDE CON L'ASSE Y DI EQUAZIONE X=0"
130 GOTO 90
135 PRINT"LA RETTA E' PARALLELA ALL'ASSE X DI EQUAZIONE Y=-C/B"
140 PRINT"CIOE' Y=";-C/B
145 GOTO 90
150 PRINT"LA RETTA E' PARALLELA ALL'ASSE Y DI EQUAZIONE X=-C/A"
155 PRINT"CIOE' X=";-C/A
160 GOTO 90
165 PRINT"LA RETTA PASSA PER L'ORIGINE"
170 PRINT"CON EQUAZIONE Y=-A/B*X"
175 PRINT"CIOE' PER X=10 Y=";-A/B*10
180 GOTO 90
185 END
```

Fig. 3 – Codifica in linguaggio BASIC *del programma della fig. 2.*

Nell'istruzione 35 vi è una **PRINT** con il messaggio che dice di introdurre i valori dei parametri A, B, C della retta, i quali vengono immessi nel computer mediante l'istruzione (INPUT A, B, C). Nelle istruzioni 5Ø e 9Ø vi sono due **PRINT**, costituite da quaranta trattini orizzontali, ottenuti battendo quaranta volte il tasto — . I trattini hanno lo scopo di delimitare sullo schermo la stampa dei risultati dell'elaborazione.

Le istruzioni 55, 6Ø, 65, 7Ø, 75 e 8Ø di diramazione condizionata pre-

vedono tutte le possibili combinazioni dei parametri A, B, C. Come si nota, in tali istruzioni vengono utilizzate relazioni logiche tipo AND (corrispondenti alla congiunzione "e"). Ad esempio, nell'istruzione 55 se A = Ø e (AND) B <>Ø e (AND) C = Ø, allora l'elaborazione salta alla linea di programma 115.

Nell'istruzione 95 è previsto un arresto temporaneo dell'elaborazione, ossia uno **STOP**, con la visualizzazione del messaggio: BREAK IN 95.

Se si desidera effettuare il calcolo di altre rette, ossia introdurre altri valori dei parametri A, B, C, occorre eseguire il comando BASIC **CONT**, che consente di riprendere da capo l'elaborazione (GOTO 3Ø).

Le istruzioni **PRINT** 1ØØ e 1Ø5 provvedono a visualizzare sullo schermo i valori di Y1 e Y2, corrispondenti alle ordinate dei punti P1 e P2 della retta.

Le rimanenti istruzioni **PRINT** consentono di visualizzare i messaggi che indicano il tipo di retta ottenuta con una particolare combinazione dei parametri A, B, C.

Terminato l'esame del listato della *fig. 3*, batta sulla tastiera il programma in questione, **procedendo con molta calma ed attenzione.**

Dopo avere immesso il programma nel computer, è opportuno listarlo battendo il comando BASIC **LIST**. Poiché tale programma è di lunghezza superiore alle righe dello schermo, dovrà listarlo prendendo in considerazione un gruppo di istruzioni per volta.

Controlli attentamente il programma listato sullo schermo con quello riportato nella *fig. 3*: se tutto è regolare, può passare ad eseguirlo battendo il comando BASIC **RUN** seguito da **RETURN** .

Sullo schermo comparirà la scritta:

```
INTRODUCI A, B, C
?
```

A questo punto provi ad inserire i seguenti, diversi valori dei parametri A, B, C.

a) ? Ø, Ø, Ø
Dovrebbe apparire sullo schermo la scritta:

```
LA RETTA NON ESISTE PERCHE' A=Ø, B=Ø, C=Ø
```

b) ? 2, Ø, −4

Dovrebbe apparire il messaggio:

```
LA RETTA E' PARALLELA ALL'ASSE Y DI EQUAZIONE X=-C/A
CIOE' X=2
```

c) ? Ø, −2, 6

Dovrebbe venire visualizzato il messaggio:

```
LA RETTA E' PARALLELA ALL'ASSE X DI EQUAZIONE Y=-C/B
CIOE' Y=3
```

d) ? 2, Ø, Ø

Dovrebbe apparire il messaggio:

```
LA RETTA COINCIDE CON L'ASSE Y DI EQUAZIONE X=Ø
```

e) ? Ø, 2, Ø

Dovrebbe apparire il messaggio:

```
LA RETTA COINCIDE CON L'ASSE X DI EQUAZIONE Y=Ø
```

f) ? 2, −4, 6

Dovrebbero apparire le coordinate dei due punti attraverso cui passa la retta di equazione:

```
X1= Ø        Y1=1.5
X2=1Ø        Y2=6.5
```

g) ? 2, −2, Ø

Dovrebbe venire visualizzato il messaggio:

```
LA RETTA PASSA PER L'ORIGINE CON EQUAZIONE Y=-A/B*X
CIOE' PER X=1Ø,  Y=1Ø
```

Il programma in questione, dopo essere stato provato e collaudato, può essere registrato con il nome "RETTA DEL PIANO" o con un altro nome a piacere (sempreché sia significativo) sul lato B della cassetta-nastro, dopo l'ultimo programma da Lei registrato.

La procedura ormai Le è familiare: si tratta di eseguire il comando fittizio

VERIFY "NOME ULTIMO"

dove per "NOME ULTIMO" si intende il nome dell'ultimo programma da Lei registrato.

In tal modo la testina di lettura/scrittura dell'unità a nastro si posizionerà esattamente dopo quest'ultimo programma, pronta per registrare il nuovo programma presente in memoria.

Dopo avere posizionato correttamente la testina del registratore, può registrare il programma, scrivendo il comando BASIC:

SAVE "RETTA DEL PIANO"

e premendo i tasti PLAY e RECORD del registratore.

2° ESERCIZIO

Vediamo ora un semplice programma, che esegue il calcolo del fattoriale di un numero.

Per fattoriale di un numero intero **n** si intende il prodotto dei numeri interi da 1 a **n**: esso viene normalmente indicato con il simbolo: **n!**

In altre parole:

$$n! = 1 \cdot 2 \cdot 3 \cdot 4 \ldots (n-1) \cdot n.$$

Ad esempio, il fattoriale del numero 5 è:

$$5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120.$$

I numeri fattoriali trovano applicazione, fra l'altro, nei calcoli statistici.

Nel calcolo del fattoriale di un numero si può facilmente superare il più grande numero ammesso dal computer.

In particolare, il supero del limite aritmetico massimo viene ottenuto eseguendo il calcolo del fattoriale di un numero superiore a 33.

In questi casi viene visualizzato sullo schermo il messaggio d'errore:

OVERFLOW
(supero di capacità)

Il flow-chart del programma in questione è riportato nella *fig. 4*.

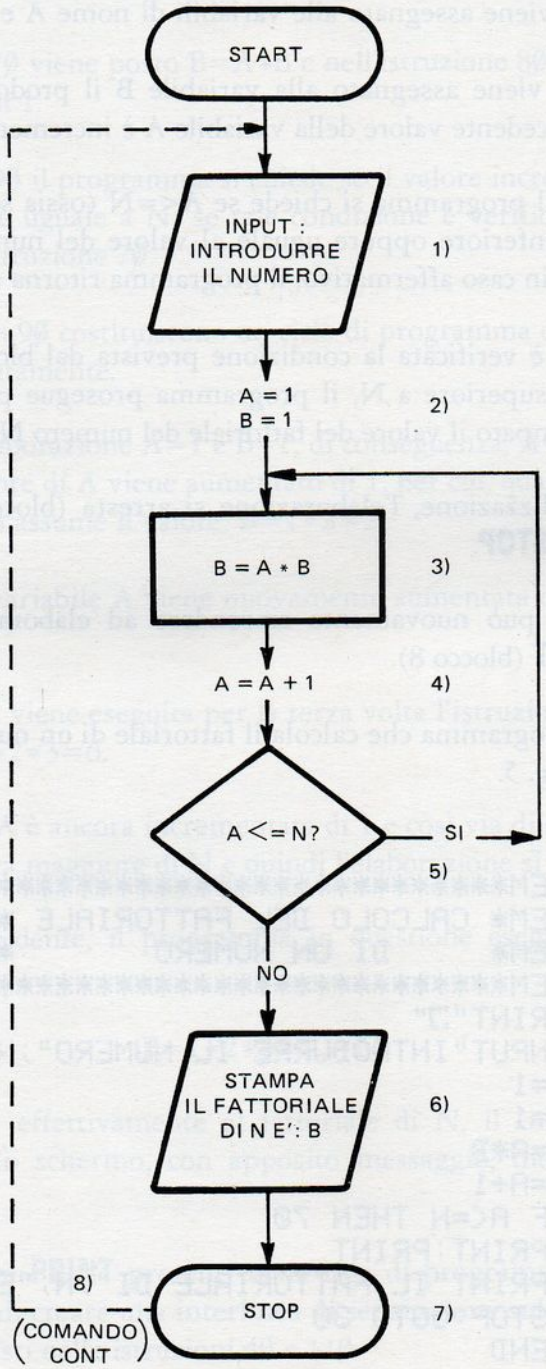


Fig. 4 - Flow-chart del programma che calcola il fattoriale di un numero.

Nel blocco 1) è presente un'istruzione **INPUT**, con visualizzazione del messaggio:

INTRODURRE IL NUMERO

la quale prevede l'immissione del numero intero N (inferiore a 34) di cui si vuole calcolare il fattoriale.

Nel blocco 2) viene assegnato alle variabili di nome A e B il numero 1.

Nel blocco 3) viene assegnato alla variabile B il prodotto A*B e nel blocco 4) il precedente valore della variabile A è incrementato di 1.

Nel blocco 5) il programma si chiede se A<=N (ossia se il valore della variabile A è inferiore oppure uguale al valore del numero N battuto sulla tastiera): in caso affermativo, il programma ritorna al blocco 3).

Se invece non è verificata la condizione prevista dal blocco 5), in altre parole se A è superiore a N, il programma prosegue per il blocco 6), dove viene stampato il valore del fattoriale del numero N.

Dopo la visualizzazione, l'elaborazione si arresta (blocco 7) mediante l'istruzione di **STOP**.

Il programma può nuovamente riprendere ad elaborare mediante il comando **CONT** (blocco 8).

Il listato del programma che calcola il fattoriale di un numero è rappresentato nella *fig. 5*.

```
10 REM*****************************
15 REM* CALCOLO DEL FATTORIALE *
20 REM*     DI UN NUMERO       *
25 REM*****************************
30 PRINT"□"
40 INPUT"INTRODURRE IL NUMERO";N
50 A=1
60 B=1
70 B=A*B
80 A=A+1
90 IF A<=N THEN 70
100 PRINT:PRINT
110 PRINT"IL FATTORIALE DI";N;"E'";B
120 STOP:GOTO 30
130 END
```

Fig. 5 - Codifica in linguaggio BASIC del programma che calcola il fattoriale di un numero.

L'istruzione 3Ø, come è noto, provvede a ripulire lo schermo ed a posizionare il cursore in alto, nell'angolo sinistro dello stesso.

L'istruzione 4Ø prevede l'immissione del numero N, di cui si desidera calcolare il fattoriale.

Nelle istruzioni 5Ø e 6Ø, come già si è visto nel flow-chart, viene assegnato alle variabili A e B il valore 1.

Nell'istruzione 7Ø viene posto B=A*B e nell'istruzione 8Ø il valore di A è incrementato di 1.

Nell'istruzione 9Ø il programma si chiede se il valore incrementato di A è minore oppure uguale a N: se tale condizione è verificata, l'elaborazione torna all'istruzione 7Ø.

Le istruzioni 7Ø÷9Ø costituiscono un ciclo di programma che ora esamineremo dettagliatamente.

All'inizio dell'elaborazione A=1 e B=1; di conseguenza, A*B=1. Successivamente il valore di A viene aumentato di 1, per cui, quando è eseguita l'istruzione 7Ø, B assume il valore: B=1*2=2.

Dopo di ciò, la variabile A viene nuovamente aumentata di 1 ed assume il valore A=3.

Di conseguenza, viene eseguita per la terza volta l'istruzione 7Ø, in base alla quale: B=1*2*3=6.

Poi la variabile A è ancora incrementata di 1 e così via di seguito, sino a quando A diventa maggiore di N e quindi l'elaborazione si ferma.

Come risulta evidente, il programma in questione esegue pertanto il calcolo:

$$B=1*2*3*4...*N$$

che corrisponde effettivamente al fattoriale di N, il cui valore viene visualizzato sullo schermo, con apposito messaggio, mediante l'istruzione 11Ø.

Le due istruzioni **PRINT** presenti nella riga di programma 1ØØ hanno invece lo scopo di creare una interlinea di separazione sullo schermo fra i messaggi previsti nelle istruzioni 4Ø e 11Ø.

Pagina mancante

PARLA BASIC

Corso teorico-pratico per corrispondenza di

ANALISI E PROGRAMMAZIONE BASIC

Scuola Radio Elettra

10126 Torino - Via Stellone 5

Tel. (011) 674432